Marco Polo

Un enfoque integrado para la resolución del problema de ordenamiento de flota y asignación de tripulación en empresas de transporte público

Pablo Banchero - Matías Prieto - Paula Riganti

Proyecto de Grado
Ingeniería en Computación
Facultad de Ingeniería,
Universidad de la República,
Montevideo – Uruguay
2013

Tutores:
Dr. Antonio Mauttone
MSc. María Urquhart

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Resumen

La problemática asociada a las empresas de transporte es un objeto de estudio interesante dado su alto grado de complejidad. En este contexto existen distintos niveles de planificación, cada uno de los cuales cuenta con sus propias características y dificultades. La etapa de planificación atacada en este proyecto es la llamada planificación operativa en la cual se deben cubrir un determinado conjunto de viajes fijos de la manera más eficiente posible en relación a los costos en los que incurren las empresas.

El problema concreto a resolver consiste en, una vez que se encuentran fijados los viajes a realizar y que se cuenta con una flota de vehículos disponibles y un conjunto de reglas laborales que regulan la labor del personal contratado, determinar la secuencia en la cual serán ejecutados los viajes de manera de que se respeten las reglas laborales existentes y de que se minimicen los costos operacionales asociados a dicha ejecución. Este problema consta de dos sub-problemas implícitos de asignación a resolver. Uno es el de asignación de flota, en el cual deben definirse los viajes que realizará cada vehículo a lo largo del día y el otro es el de asignación de tripulación, para la operación de cada uno de los vehículos, en el cual deben definirse como estarán compuestas las jornadas laborales de cada trabajador. El resultado de estas asignaciones es a lo que se le llama libro de servicios y su planificación conjunta es lo que se quiere optimizar en el presente proyecto.

Dado que cada una de las asignaciones a resolver persigue sus propios objetivos y cuenta a su vez con restricciones particulares, se decide abordar un enfoque de resolución integrado. El mismo pretende tomar en cuenta las características de ambos sub-problemas en simultáneo para que la resolución de cada uno de ellos coopere con la otra, con el propósito de encontrar una mejor solución al problema. En Marco Polo se plantea la utilización de la metaheurística Algoritmos Genéticos para diseñar el algoritmo de resolución del problema. Además se presentan el diseño y la implementación de un sistema modular, que brinda soporte a dicho algoritmo, cuyo núcleo es un grafo que maneja las dimensiones espacio y tiempo. En este sistema cada módulo se ocupa de tareas concretas, como ser la generación de soluciones iniciales y la gestión de las reglas laborales, entre otras, con el fin de atacar cada tarea de forma aislada y así simplificar el funcionamiento y el entendimiento del sistema en su conjunto.

Se utilizan dos casos de estudio, uno real y otro ficticio existente en la literatura; estos se comparan con los correspondientes resultados de dos proyectos antecesores y con la solución manual obtenida por la empresa a la cual pertenece el caso real, constatando que, en varios de los escenarios planteados, Marco Polo mejora los resultados existentes.

Palabras clave: Transporte público, Ordenamiento de Vehículos, Asignación de Tripulación, Reglas Laborales, Metaheurísticas, Algoritmos Genéticos, Enfoque Integrado, Optimización combinatoria.

Abstract

The problems associated with public transit companies are an interesting object of study because of its high degree of complexity. In this context there are different levels of planning, each of which has its own characteristics and difficulties. The planning stage attacked in this project is called Operational Planning and aims at covering a given set of fixed travels as efficiently as possible in relation to the costs that companies incur.

The specific problem to be solved is, once you have the set of travels to perform, a fleet of vehicles available, and a set of labor rules governing the work of the contract staff, determine the sequence in which the travels will be executed in order to respect the operational constraints and minimize the associated costs. This problem consists of two implicit scheduling sub-problems to be resolved. One is the bus scheduling, which must be defined to perform each vehicle travel along the day, and the other is the crew scheduling, which must define the labor of each worker. The result of these two assignments, the bus and crew scheduling, is what is called Service Book and its joint planning is what is to be optimized in this project.

Since each of the assignments pursues its own objectives and its own particular constraints, we decided to address an integrated solving approach. This approach is intended to take into account the characteristics of both sub-problems simultaneously resolving them so that each cooperates with the other, in order to find a better solution to the problem. Marco Polo describes the use of the Genetic Algorithms metaheuristic to design an algorithm to solve the problem. Besides it incorporates the design and implementation of a modular system that supports the algorithm, whose core is a graph which handles the space and time dimensions. In this system each module is responsible for specific tasks, such as initial solution generation and management of labor rules, among others, in order to attack each task in isolation to simplify the operation and understanding of the whole system.

Using a benchmark test case and a real one, comparing the results with the corresponding results of two predecessor projects and the manual solution obtained by the company which owns the real case, it concludes that, in several of the proposed scenarios, Marco Polo improves the existing results.

Keywords: Public Transportation, Vehicle Scheduling, Crew Scheduling, Work Rules, Metaheuristics, Genetic Algorithms, Integrated Approach, Combinatorial Optimization.

Contenido

Capítulo 1: Introducción

1.1 Presentación del problema

Hoy en día el transporte público colectivo es un medio de locomoción utilizado por una inmensa cantidad de personas debido a sus beneficios entre los cuales se destacan su alcance económico, el mantener alejados a las personas del estrés que genera el automóvil y el de ser, en definitiva, la alternativa más ecológica y solidaria para muchos de los desplazamientos que se hacen dentro del casco urbano, evitando así el uso sistemático y masivo del vehículo privado que colapsa las ciudades y las hace más sucias y más ruidosas.

El problema del transporte público (1) radica en poder satisfacer la demanda de los usuarios de la manera más eficaz y eficiente posible. Debido a las diversas áreas y aspectos que están involucrados en la problemática aquí tratada, el mismo puede ser dividido en los siguientes tres sub-problemas:

- *Planeamiento Estratégico:* Aquí el problema a resolver consiste en diseñar rutas y redes de transporte público. Se toman en cuenta, como datos fundamentales, la demanda de los usuarios de viajes de un punto a otro y el diseño de la ciudad (rutas, calles, avenidas, etc.). Las líneas obtenidas como resultado de la resolución de este sub-problema son ofertadas en una licitación la cual otorga al ganador de la misma la concesión del conjunto de líneas.

- *Planeamiento Táctico:* Al momento de enfrentar esta etapa ya se consideran las rutas como determinadas y lo que se debe obtener es la frecuencia con la cual se ofrecerá cada una de ellas mediante la generación de una tabla de horarios (*timetable*) la cual es el producto final de ésta etapa. El dato más relevante para ésta etapa es la demanda de los usuarios.

- *Planeamiento Operativo:* Este problema se orienta en la búsqueda de asignación de los recursos con los que cuentan las compañías de ómnibus (flota y tripulación) a los viajes programados en la etapa de Planeamiento Táctico, con fuerte énfasis en la minimización de costos.

El presente proyecto está dedicado pura y exclusivamente al estudio y resolución del Planeamiento Operativo. Este estudio se realiza en el marco de un convenio realizado entre el Departamento de Investigación Operativa de la Facultad de Ingeniería de la Universidad de la República y la Compañía de Ómnibus de Pando S.A. el cual prevé la realización de estudios para optimizar la operativa de dicha empresa. A partir de este convenio surgen, entre otros, el proyecto de grado del año 2009, en adelante el PG42 (2), el cual se puede considerar un predecesor del presente trabajo. En él se realiza el estudio del estado del arte y se propone una solución al problema de ordenamiento de flota y asignación de tripulación buscando minimizar costos operativos, considerando una realidad modelada y definida a través de parámetros, restricciones y reglas. En el Capítulo 4 se presenta un análisis de dicho proyecto.

1.2 Motivación

El problema de optimización planteado se agrupa dentro de la rama de optimización combinatoria. En este tipo de problemas se busca encontrar, dentro de un conjunto de posibles soluciones discreto, la mejor solución que es aquella que minimice una función de costos previamente definida. Existen técnicas de resolución basadas en búsquedas exhaustivas que son adecuadas para la resolución de este tipo de problemas cuando su complejidad es baja o media. A medida que la complejidad del espacio de búsqueda aumenta se torna imposible analizar y evaluar todas las posibles soluciones ya que el número de combinaciones crece exponencialmente con el tamaño del problema. Este aumento también viene acompañado a un aumento en el costo de ejecución de este tipo de algoritmos lo cual no es deseable dado que en este tipo de problemas suelen existir restricciones de tiempo para el proceso de toma de decisiones. En estos casos se utilizan técnicas que, en lugar de perseguir la solución óptima y además el mejor tiempo de ejecución, sacrifican parte de ambos objetivos, buscando una solución de calidad pero no necesariamente la óptima, a cambio de tiempos de ejecución razonables. Este tipo de técnicas se dividen en dos grandes grupos (3): las técnicas Heurísticas, en las cuales se explota el conocimiento del dominio de aplicación para encontrar soluciones de buena calidad a problemas combinatorios complejos basándose en procedimientos conceptualmente simples, y las Metaheurísticas, que consisten de algoritmos de propósito general que guían el proceso de búsqueda combinando y subordinando distintos métodos heurísticos, implementando así una estrategia de más alto nivel. En el Capítulo 2 se profundiza en la resolución de problemas de optimización.

El problema de planificación de libros de servicio es un problema de optimización combinatoria complejo por lo que requiere el uso de técnicas avanzadas. Un libro de servicios es, en términos generales, el resultado de las asignaciones de los viajes, previamente definidos, a la flota de vehículos y a la tripulación existente. En el presente proyecto se decide usar como estrategia de resolución un Algoritmo Genético (4). El problema es atacado con el fin de mejorar la utilización de los recursos con los cuales cuentan las compañías de ómnibus. Al optimizar sus recursos se obtendrán beneficios económicos tanto de manera directa para las propias compañías así como también para los usuarios finales de los servicios que estas brindan.

Por otro lado los problemas de asignación de flota y tripulación cuya solución, como ya se mencionó, son los libros de servicios, han sido abordados a lo largo del tiempo mayoritariamente con enfoques secuenciales (5) dada la complejidad que implica cada uno de los problemas en sí mismos. Esto significa que se resuelven, por un lado, la asignación de la flota de vehículos a los viajes y luego, sobre este problema ya resuelto, se define la asignación de la tripulación. Esto resulta en resolver dos sub-problemas consecutivos, cada uno de ellos, menos complejo que el problema original.

En el año 1983 aparece el primer planteo de resolución al problema integrado (6). Más recientemente aparecen los trabajos de Freling (7), Gaffi y Nonato (8) y Huisman (9) utilizando técnicas heurísticas y relajaciones de Lagrange y en el 2008 surge la realizada por Jin-Kao Hao y Laurent Benoit (10) donde su resolución se basa en las técnicas GRASP (*Greedy Randomized Adaptive Search Procedure)* e ILS (*Improvement*

by Local Search). En el año 2010 el proyecto predecesor al presente trabajo, el PG42 (2), plantea una aproximación a la resolución del problema integrado con la técnica de Algoritmos Genéticos. La motivación del presente proyecto es mejorar los resultados previamente obtenidos enfocándonos en la resolución del caso COPSA. Se plantea el análisis del proyecto predecesor (Capítulo 4) para determinar cuáles de sus aspectos son mejorables así como aprovechar las características positivas del mismo.

1.3 Objetivos

El presente proyecto tiene dos grandes objetivos. El primero de ellos radica en obtener mejoras de algoritmos para la optimización de libros de servicio de empresas de transporte público colectivo urbano y suburbano. Para evaluar el logro o no de este objetivo se harán comparaciones entre los resultados obtenidos en el presente proyecto con los obtenidos por el PG42, el proyecto desarrollado por el Departamento de Investigación Operativa del Instituto de Computación de la Facultad de Ingeniería (en adelante Sistema IO) y con los resultados obtenidos por un planificador experto en la materia. El segundo objetivo planteado es el de atacar el problema desde un enfoque integrado resolviendo en simultáneo ambas asignaciones. Esto es que, tanto la tarea de asignación de flota como la de asignación de tripulación a los viajes que fueron definidos en la etapa de planificación previa, se logren resolver de manera simultánea y cooperativa en lugar de plantear una solución en la cual las asignaciones sean resueltas de modo secuencial. Con esto se busca que la resolución de cada uno de los problemas retroalimente a la otra de manera que en todo momento se estén teniendo en cuenta ambos objetivos. La resolución secuencial, en cambio, puede llevar a que la primera asignación se resuelva de manera óptima pero luego la segunda asignación quede atada a los resultados de la primera viéndose afectada la calidad de su resolución.

1.4 Resultados esperados

Con respecto al primer objetivo planteado se espera obtener resultados que supongan un uso eficiente de los vehículos que componen la flota así como también de los recursos humanos disponibles. Esto se debe corresponder con la obtención de datos numéricos que mejoren los resultados obtenidos por los proyectos contra los cuales haremos las comparativas y que por otro lado le resulte rentable a la empresa mejorando, o al menos igualando, los resultados obtenidos mediante la resolución manual que actualmente se lleva a cabo. La evaluación se realizara sobre dos escenarios que presentan dimensiones disímiles. Para el escenario de prueba elaborado en el Departamento de Investigación Operativa se cuenta con dos soluciones contra las cuales se pueden comparar los resultados del presente proyecto: la obtenida por el Sistema IO y la del PG42. El segundo escenario es dado por la empresa de transporte público y además de contar con soluciones del Sistema IO y el PG42 se dispone de la solución construida por un planificador experto.

Para esto contamos con dos casos de estudio elaborados, uno de ellos, por el Departamento de Investigación Operativa para el Sistema IO y, el otro, por un planificador experto el cual se corresponde con un caso real de una empresa operando en el medio local.

Con respecto al segundo objetivo se desea poder llegar a generar un aporte en el campo científico en cuanto a estudiar la resolución de este problema particular, de asignación de flota y tripulación a viajes predefinidos, desde un enfoque integrado utilizando como estrategia de resolución un Algoritmo Genético.

1.5 Organización general del documento

El presente documento está organizado en capítulos y dentro de éstos, en secciones y sub-secciones. El primer capítulo consta básicamente de una introducción, definiendo el problema a resolver, planteando la motivación del trabajo y los objetivos planteados junto con los resultados esperados. El segundo y tercer capítulo presentan el marco conceptual del problema a resolver y las definiciones y especificaciones de los conceptos claves usados a lo largo del documento. Ambos capítulos serán fundamentales para un buen entendimiento del resto del documento. Los siguientes dos capítulos contienen la documentación de la parte central del trabajo incluyendo análisis, diseño, arquitectura y decisiones de implementación. Los últimos dos capítulos contienen las pruebas y comparativas realizadas, los resultados obtenidos y las conclusiones del proyecto. Se incluye como anexo un manual de instalación así como documentación detallada de los módulos desarrollados.

Capítulo 2: Marco conceptual

En este capítulo se introducen los problemas de optimización combinatoria junto con una reseña que enumera sus distintos métodos de resolución (las principales definiciones referidas a este tema son tomadas de (3), (11), (12) y (13)) para luego ahondar en el funcionamiento de los Algoritmos Genéticos (4), (13) técnica con la cual se trabaja en el presente proyecto.

2.1 Problemas de optimización combinatoria

Los problemas de optimización combinatoria generalmente vienen asociados a una elevada complejidad computacional y, en este caso, son una clase particular de problemas NP. Su objetivo es hallar la o las soluciones óptimas de un problema cuyo dominio es discreto, en base a una función objetivo determinada. Mientras el tamaño del problema es pequeño se puede resolver con métodos exhaustivos de búsquedas de soluciones pero, a medida que el tamaño del problema aumenta, las técnicas exhaustivas se tornan inmanejables en cuanto al tiempo estimado para su resolución. En estos casos se hace necesario utilizar algoritmos de resolución más eficientes teniendo siempre en cuenta que la complejidad del mismo se verá incrementada a medida que crece el tamaño de la entrada.

El planteo de un problema de optimización combinatoria tiene la siguiente forma:

$$\min f(x) \quad tal\ que\ x \in D$$

donde:

- $x = (x_1, x_2, x_3, \ldots x_n)$ *son las variables del problema*
- $f(x)$ *es la función objetivo*
- $\forall\, i, x_i \in D_i$ *dominio de la variable, el cual es un conjunto discreto* $(finito\ o\ infinito)$
- $X = D_1 x\, D_2 x \ldots D_n$ *espacio de soluciones discreto*
- $D \subset X$ *espacio de soluciones factibles*

Un problema de optimización combinatoria tiene la restricción de que las variables de decisión pertenecen a dominios discretos y que las soluciones se representan como permutaciones o combinaciones. Este tipo de problemas puede ser especificado como un conjunto de instancias, donde, cada una de ellas está asociada con una solución del espacio discreto. Dentro del espacio discreto está el espacio factible que es el que contiene el conjunto de soluciones que satisface las restricciones del problema. Estos problemas se pueden ver como problemas de búsqueda en donde, dado un conjunto finito de elementos, se debe encontrar uno que cumpla un determinado conjunto de propiedades.

2.1.1 Técnicas de resolución de problemas de optimización

- *Técnicas analíticas*: solo son aplicables a problemas con un nivel bajo de generalidad y con restricciones que tengan menor o igual número de variables que la función objetivo o directamente sin restricciones

- *Métodos exactos*: basados en técnicas de enumeración total o parcial como Branch & Bound o Backtracking sin poda respectivamente, o en algoritmos de programación matemática como el Método Simplex o el Método del Punto Interior

- *Métodos de aproximación*: encuentran una solución con un error máximo conocido a priori respecto del óptimo. Se basan en probabilidades a lo largo del tiempo o en el estado en el que terminarán los acontecimientos en el sentido de una inducción que permite establecer una verdad con mayor índice de probabilidad que las demás. Ejemplos de estos métodos son los métodos markovianos o de inferencia probabilística.

- *Métodos aleatorios*: estos métodos encuentran con cierta probabilidad una solución con un error máximo dado. La probabilidad de encontrar una solución correcta aumenta con el tiempo empleado en obtenerla y el número de muestreos utilizado. El objetivo es minimizar la probabilidad de no encontrar la solución, tomando decisiones aleatorias con inteligencia, pero minimizando también el tiempo de ejecución al aplicarse sobre el espacio de información aleatoria. Ejemplos de este tipo de métodos son el método de Monte Carlo y método de Las Vegas.

- *Técnicas Heurísticas*: estas técnicas se basan en procedimientos conceptualmente simples y encuentran soluciones de buena calidad a problemas difíciles de un modo sencillo y eficiente aunque sin garantizar la optimalidad de las mismas. Existe una variada gama de técnicas heurísticas siendo las más útiles generalmente la llamadas heurísticas específicas que incorporan conocimiento del problema de optimización a resolver. Las técnicas heurísticas pueden dividirse en: (a) heurísticas constructivas, las cuales suelen ser métodos rápidos que van construyendo una solución, partiendo de una inicialmente vacía, incorporando componentes hasta llegar a obtener una solución completa, y (b) métodos de búsqueda local, los cuales se basan en el concepto de vecindario, parten de una solución ya completa y recorren el vecindario actual (el vecindario va cambiando en las sucesivas iteraciones de acuerdo a mecanismos internos del algoritmo) quedándose cada vez con el mejor individuo hasta encontrar un óptimo local.

- *Técnicas Metaheurísticas*: son algoritmos no exactos cuya idea es combinar distintos métodos heurísticos a un nivel más alto de abstracción para conseguir resolver un tipo de problema computacional general realizando una exploración del espacio de búsqueda de forma eficiente y efectiva. Son técnicas generales que guían el proceso de búsqueda y tienen como objetivo encontrar

rápidamente soluciones factibles y de buena calidad. Las metaheurísticas hacen uso de conocimiento del problema que se trata de resolver usando métodos heurísticos subordinados que son controlados por una estrategia de más alto nivel. Las técnicas metaheuristicas pueden dividirse en: (a) metaheurísticas basadas en trayectoria, las cuales parten de una solución y, mediante la exploración del vecindario, van actualizando la solución actual formando una trayectoria, y (b) metaheurísticas basadas en población, las cuales se caracterizan por trabajar con un conjunto de soluciones, denominado población, en cada iteración.

Las técnicas heurísticas y metaheurísticas constituyen alternativas para hallar soluciones de buena calidad en tiempos razonables a problemas de optimización combinatoria complejos. En este tipo de técnicas es importante manejar un correcto equilibrio entre las técnicas de exploración y explotación del espacio de búsqueda.

- *explotación*: la explotación guía la búsqueda teniendo en cuenta la información obtenida de puntos del espacio de búsqueda previamente visitados para determinar los puntos que conviene visitar a continuación. Involucra pequeños saltos en el espacio de búsqueda y es un mecanismo provechoso para que un algoritmo encuentre óptimos locales.

- *exploración*: la exploración se encarga de visitar nuevas regiones del espacio de búsqueda para tratar de encontrar soluciones potencialmente mejores. Involucra grandes saltos en el espacio de búsqueda y es un mecanismo útil para evitar una convergencia prematura.

En el presente proyecto se decide abordar el problema con la técnica metaheurística, basada en población, de Algoritmos Genéticos. Este tipo de técnicas trabajan sobre una población de soluciones que evoluciona mediante mecanismos de selección y construcción de nuevas soluciones por recombinación de características de las soluciones originales. A continuación se presentan sus elementos fundamentales y se explica su funcionamiento.

2.2 Algoritmos genéticos

Los algoritmos genéticos están inspirados en la teoría de la evolución natural de los seres vivos para resolver problemas de optimización y búsqueda. Las primeras referencias a esta teoría surgen en el texto de Holland del año 1975 (14), aunque uno de los primeros textos en los que se presenta una exposición exhaustiva, con aplicaciones a problemas reales, es el texto de Goldberg del año 1989 (4). Esta familia de técnicas sigue un proceso iterativo que opera sobre una población de individuos que representan soluciones codificadas de alguna manera, las cuales son generadas inicialmente con alguna técnica generalmente aleatoria que puede ser o no constructiva. La idea del algoritmo es que estos individuos interactúen entre sí con el objetivo de producir, en cada generación, mejores soluciones al problema. El esquema general de un algoritmo genético se compone de cuatro fases principales: evaluación, selección, reproducción y reemplazo. El proceso completo es repetido hasta que se cumpla un cierto criterio de terminación, normalmente después de un número dado

de iteraciones. En la fase de evaluación se asigna a cada individuo un valor de aptitud para luego, en la fase de selección, seleccionar en base a dicho valor de aptitud los individuos de la población actual que serán recombinados en la fase de reproducción. Los individuos resultantes de la recombinación se alteran mediante un operador de mutación. Finalmente, a partir de la población actual y/o los individuos generados, de acuerdo a su valor de aptitud o fitness, se forma la nueva población, dando paso a la siguiente generación del algoritmo.

El esquema general de un algoritmo genético es el siguiente:

```
InicializarAleatoriamente(P(0));
generacion = 0;
mientras (no CriterioParada) hacer
        Evaluar(P(generacion));
        Padres = Seleccionar(P(generacion));
        Hijos = Aplicar Operadores Evolutivos(Padres);
        NuevaPoblacion = Remplazar(Hijos,P(generacion));
        generacion++;
        P(generacion) = NuevaPoblacion;
fin
retornar Mejor Solución Encontrada
```

Figura 2.1: Esquema de un algoritmo genético

Un algoritmo genético tiene cinco componentes básicos:

2.2.1 Representación de la solución

La representación es a la solución lo que el *genotipo* es al *fenotipo* si queremos expresarnos usando los términos biológicos propios de la teoría que hay detrás de los algoritmos genéticos. Se debe definir un proceso o función de codificación que permita pasar del fenotipo al genotipo estableciendo una correspondencia entre uno y otro de manera que se pueda codificar cualquier solución perteneciente al espacio de soluciones. Debe existir también una función que realice el proceso inverso, el de decodificación, de manera de saber interpretar luego una solución codificada. La complejidad de estas funciones y el tipo de representación dependerán de las características del problema y de cuáles y como sean las variables a codificar.

Los términos utilizados para nombrar los distintos componentes de la solución tienen origen en los conceptos biológicos en los que se basan los algoritmos genéticos. A la estructura usada para almacenar la solución codificada (el genotipo) se la denomina *cromosoma*. La representación más simple tradicionalmente usada es una cadena binaria. A cada subestructura dentro del cromosoma con información genética se le denomina *gen* y a la información almacenada dentro del gen se le llama *alelo*. En el caso de la cadena binaria un gen sería una sub-cadena de la cadena original compuesta por una o más posiciones y el alelo sería la información de ceros y unos contenida el gen.

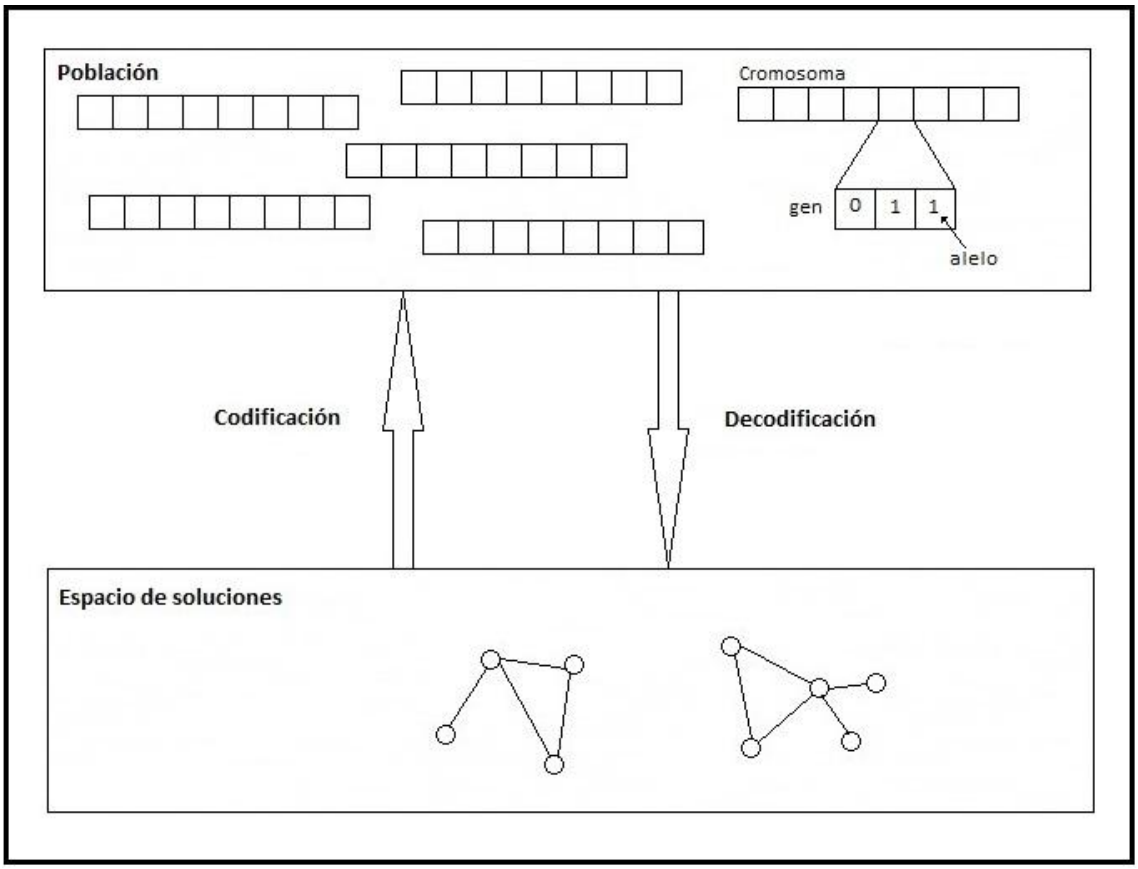

Figura 2.2: Codificación y decodificación de soluciones

2.2.2 Población inicial

Se debe definir un procedimiento, el cual puede ser aleatorio o utilizar algún método heurístico o constructivo, para construir los individuos que formarán parte de la primera generación del algoritmo. Al ser la idea de los algoritmos genéticos la de que prevalezcan las mejores características de sus individuos a lo largo de las sucesivas generaciones, se debe contar con la mayor variedad y diversidad posible al comienzo de la ejecución. En la sección 5.4 se verá que el único factor que aportará diversidad extra a la población será el operador de mutación el cual, a su vez, no debe ser aplicado de manera indiscriminada. El resto de las características tendrán que surgir a partir de combinaciones de propiedades existentes en individuos de la población inicial. Por lo tanto si contamos con una población inicial pobre en variedad, esto es, con individuos muy similares entre sí, corremos el riesgo de provocar una convergencia temprana hacia un individuo con características muy similares también a las de los individuos iniciales. Es importante entonces garantizar que, dentro de la población inicial, exista la diversidad suficiente como para evitar la convergencia prematura.

2.2.3 Función de fitness

La función de fitness o de aptitud es la función encargada de evaluar un individuo y retornar como resultado un valor que indica que tan apto es el individuo para la resolución del problema. Esta función guía el mecanismo de búsqueda a través del

operador de selección y su influencia es fundamental para determinar los individuos candidatos a sobrevivir.

Es importante lograr construir una función de fitness adecuada la cual dependerá siempre del problema a resolver y del criterio de optimización. Esta función puede tener en cuenta las restricciones del problema y además es posible definirla de manera que cambie dinámicamente a medida que el algoritmo procede.

2.2.4 Operadores evolutivos

En cada generación se le aplican, a una parte de los individuos de la población, los operadores evolutivos. Estos operadores actúan sobre los genotipos de los individuos de manera probabilística. Su cometido es hacer que los individuos se vayan modificando con el correr de las iteraciones emulando la evolución natural con la finalidad de que el algoritmo converja a la mejor solución.

2.2.4.1 Selección

El operador de selección utiliza un mecanismo de muestreo para seleccionar los cromosomas que participarán con el rol de padres en la próxima iteración para crear la siguiente generación. El objetivo principal de este operador es mantener las características de aquellos individuos mejor adaptados. Existen distintos métodos de muestreo. En la técnica de muestreo estocástico se asigna una probabilidad a cada individuo, basándose en su valor de aptitud o fitness. La implementación más usada de muestreo estocástico es la *selección proporcional o rueda de ruleta*. Por otro lado están las técnicas de muestreo determinístico en donde se ordenan los individuos según aptitud y se seleccionan los mejores como padres. Un ejemplo de operador de selección que utiliza muestreo determinístico sería la técnica de *selección elitista* la cual asegura la permanencia de los mejores individuos. Luego están también las técnicas de muestreo mixto que incluyen características tanto deterministas como aleatorias. Un ejemplo sería la *selección por torneo* en la cual se selecciona el o los mejores individuos de un conjunto compuesto por individuos seleccionados previamente de manera aleatoria.

Este operador trae asociado el concepto de *presión selectiva* sobre los individuos mejor adaptados el cual, en definitiva, determina si predominará la exploración o la explotación del espacio de búsqueda. Si el método elegido tiene una alta presión de selección se centrará la búsqueda de las soluciones en un entorno cercano a los mejores individuos reduciéndose de esta manera la diversidad de información genética. Si en cambio tiene una baja o moderada presión de selección el algoritmo tenderá a explorar nuevas secciones del espacio de búsqueda y de esta manera se mantiene o incluso puede incrementarse la diversidad. Es importante tener en cuenta que la selección, al igual que el concepto de presión selectiva, está fuertemente ligada con la función de fitness y su funcionamiento depende de una correcta definición de dicha función.

2.2.4.2 Cruzamiento

El operador de cruzamiento tiene como objetivo fundamental garantizar la explotación de buenas secciones del espacio de búsqueda. La idea es básicamente combinar partes del individuo padre y partes del individuo madre para obtener nuevos individuos los cuales son considerados los descendientes de los primeros. Existen distintas implementaciones de este operador entre las cuales las más sencillas son las de cruzamiento en uno o varios puntos o el cruzamiento uniforme en el cual se definen distintos puntos de intercambio fijos. Luego, para codificaciones más complejas, se deben implementar cruzamientos específicos dependientes del problema.

2.2.4.3 Mutación

El operador de mutación tiene como objetivo fundamental introducir diversidad a los individuos y de esta manera explorar nuevas secciones del espacio de búsqueda. Este operador se aplica sobre un solo individuo y su resultado es el individuo con la introducción de una modificación producto de la aleatoriedad. La técnica más sencilla de mutación es la de inversión de valor de un alelo. Luego, al igual que en el caso del operador de cruzamiento, se deberán implementar operadores específicos para codificaciones complejas de acuerdo a las características del problema a resolver.

Luego de aplicados los operadores, los nuevos individuos se incorporan a la población, de acuerdo al mecanismo de reemplazo establecido que se debe encargar de realizar el recambio generacional, y continúa la evolución con una nueva iteración.

2.2.5 Configuración paramétrica

Este componente refiere a la calibración previa que se debe hacer de los distintos parámetros involucrados para obtener los valores con los cuales el algoritmo muestra un mejor desempeño. En esta instancia se evalúa un conjunto de configuraciones paramétricas haciendo un análisis experimental sobre instancias o casos de prueba que, generalmente, son distintos y más pequeños a las instancias o casos de estudio de validación. Los parámetros que deben estudiarse son el tamaño de la población y las probabilidades de cruzamiento y mutación. También se puede incluir la calibración de otros parámetros que se consideren necesarios como ser parámetros de operadores implementados específicamente para determinado tipo de problema y hasta hacer una evaluación de diferentes alternativas para el algoritmo como ser un estudio comparativo de operadores para determinar, por ejemplo, el tipo de selección a usar.

Capítulo 3: Conceptos generales

Los siguientes son conceptos y definiciones que se consideran claves para el entendimiento del resto del documento. En primer lugar se presentan los que aplican al problema y que a su vez refieren a la realidad. Luego se introducen definiciones de términos relativos al método propuesto de solución.

3.1 Conceptos relacionados a la realidad del problema

3.1.1 Viaje

El concepto base del problema es el de viaje. Un viaje es la acción que realiza un ómnibus partiendo en un determinado horario desde un lugar geográfico y arribando en un momento posterior a otro lugar geográfico distinto.

Para que un viaje quede perfectamente determinado se necesitan los siguientes datos: lugar de partida, hora de partida, lugar de arribo y hora de arribo. Un ejemplo podría ser el siguiente:

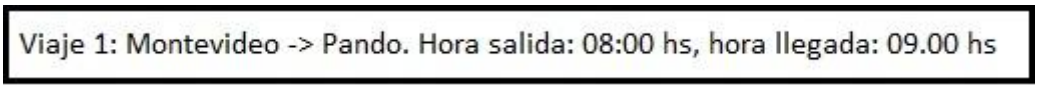

Figura 3.1: Viaje de Montevideo a Pando

De este concepto se derivan distintas categorías o tipos que agrupan a los viajes según sus características principales.

3.1.1.1 Tipo de viajes

- *Viaje Activo:* los viajes activos son aquellos viajes que la empresa ofrece a sus clientes los cuales fueron definidos en la etapa de Planeamiento Táctico. Estos viajes parten de un lugar geográfico origen a una hora dada y luego llegan a un lugar geográfico destino a otra hora que dependerá de la duración del viaje. Este tipo de viajes deben cumplirse obligatoriamente

- *Viaje Expreso:* un viaje expreso es aquel viaje que se hace entre dos lugares geográficos distintos sin llevar pasaje con el fin de trasladar un ómnibus hacia el destino para que, luego, este pueda ser utilizado en un próximo viaje activo que parta desde dicho lugar. Este tipo de viajes no forma parte de los viajes que se deben cubrir de forma obligatoria.

- *Viaje Espera:* los viajes espera son viajes en el tiempo desde un lugar geográfico en un determinado momento hacia el mismo lugar geográfico pero en un instante de tiempo posterior, sin realizar movimiento geográfico alguno, lo cual es equivalente a decir que el ómnibus que está realizando un determinado viaje espera permanecerá en un lugar fijo durante cierto periodo de tiempo. Al igual que en el caso de los viajes expresos, este tipo de viajes tampoco forma parte de los viajes que deben cubrirse obligatoriamente.

3.1.1.2 Viajes Compatibles

La compatibilidad es una relación que se define entre dos viajes, la cual toma en cuenta por un lado los horarios de comienzo y fin de cada uno y por otro los lugares geográficos de partida y arribo en términos del tiempo necesario para llegar de uno al otro. Se define que dos viajes son compatibles si y solo si pueden ser realizados consecutivamente por un mismo vehículo.

Para formalizar el concepto definimos:

- $\boldsymbol{hc_i}$ $hora\ de\ comienzo\ del\ viaje\ i$
- $\boldsymbol{hf_i}$ $hora\ de\ fin\ del\ viaje\ i$
- $\boldsymbol{t_{ij}^{d\rightarrow o}}$ $tiempo\ que\ toma\ ir\ del\ lugar\ destino\ del\ viaje\ i\ al\ origen\ del\ viaje\ j$

Dados dos viajes A y B diremos que el viaje A es compatible con el viaje B sii se cumple que:

$$\boldsymbol{hf_A + t_{AB}^{d\rightarrow o} \leq hc_B}$$

3.1.1.3 Viajes Intercambiables

La relación de viajes intercambiables se define entre dos viajes y, al igual que con los viajes compatibles, se toman en cuenta los horarios de comienzo y fin de cada uno y los lugares geográficos de partida y arribo. Dos viajes son intercambiables si y solo si es posible que entre ellos se intercambien los lugares y horarios de destino de manera que cada uno logre cubrir el nuevo viaje asignado a tiempo.

Para formalizar el concepto tomamos las definiciones anteriores y, de manera análoga a $\boldsymbol{t_{ij}^{d\rightarrow o}}$, definimos:

- $\boldsymbol{t_{ij}^{o\rightarrow d}}$ $tiempo\ que\ toma\ ir\ del\ lugar\ origen\ del\ viaje\ i\ al\ destino\ del\ viaje\ j$

Dados dos viajes A y B diremos que el viaje A es intercambiable con el viaje B sii se cumple simultáneamente que:

$$\begin{cases} \boldsymbol{hc_A + t_{AB}^{o\rightarrow d} \leq hf_B} \\ \boldsymbol{hc_B + t_{BA}^{o\rightarrow d} \leq hf_A} \end{cases}$$

3.1.2 Turno

Debido a que uno de los recursos con los cuales cuentan las empresas son los recursos humanos surge el concepto turno. De manera abstracta, decimos que un turno describe la jornada laboral de un trabajador. La jornada laboral estará compuesta por una sucesión de actividades que el trabajador debe realizar, las cuales incluyen tanto al trabajo propiamente dicho como a los descansos. Estas actividades tienen su

correspondencia con los distintos tipos de viajes siendo, por un lado, las actividades de trabajo correspondientes a los viajes activos y expresos y, por otro, las actividades de descanso correspondientes a los viajes espera. Los turnos están gobernados por lo que se denominan reglas laborales. Estas reglas validan la secuencia de actividades a realizar por el trabajador. Las reglas laborales son producto de acuerdos y leyes que le imponen al trabajador ciertas obligaciones y le conceden ciertos derechos.

3.1.2.1 Regla Laboral

Una regla laboral es cualquier restricción impuesta sobre la jornada laboral de un trabajador. Una regla laboral puede expresar restricciones de diversas índoles como ser: duraciones máximas o mínimas para la jornada completa, cantidad de descansos permitidos u obligatorios, duración y frecuencia de los mismos. Las reglas laborales son parte de la entrada del problema y son un factor fundamental para la función de búsqueda. Ejemplos de reglas laborales pueden ser:

- El jornal debe contener al menos dos descansos de 20 minutos cada uno.
- La duración máxima del jornal debe ser 10 horas.
- La frecuencia entre descansos debe ser superior a 180 minutos.
- Un conductor no puede permanecer más de 4 horas sin un descanso.

3.1.2.2 Microturno

Un microturno es un turno, relativamente corto, que cumple un conjunto de reglas más holgadas y menos restrictivas que las reglas que deben cumplir los turnos regulares.

3.1.2.3 Turno Factible

Definimos como turno factible a todo aquel que cumpla con las reglas laborales definidas en el problema.

3.1.3 Vehículo

Otro de los recursos principales con los que cuenta la empresa son las unidades mecánicas, esto es, los ómnibus. Este recurso se representa en el presente proyecto con el concepto vehículo. A un vehículo se le asignan una secuencia ordenada de viajes que deberán ser ejecutados por la tripulación asignada al mismo en el correr del día. Cada vehículo tiene una categoría asignada la cual depende de sus características, como ser, modelo del vehículo, comodidades, capacidad, antigüedad, etc. Luego cada viaje podrá requerir un vehículo de determinada categoría como condición para ser llevado a cabo. Los vehículos se almacenan en depósitos desde donde parten para realizar el primer viaje de la jornada y al cual deben retornar luego de finalizar el último viaje asignado.

3.1.4 Recorrido

Un recorrido es una secuencia de viajes ordenada ascendentemente según el horario de partida. Conceptualmente se puede hablar de recorrido tanto cuando queremos

describir los viajes que debe realizar un vehículo a lo largo del día como también cuando queremos hacer referencia a los viajes que debe realizar un turno.

3.1.4.1 Recorrido factible

Definimos a continuación la propiedad de factibilidad sobre un recorrido. Decimos que un recorrido es factible si y solo si sus viajes pueden ser ejecutados secuencialmente y no hay "huecos horarios", es decir, que inmediatamente cuando termina uno comienza el siguiente en el mismo lugar geográfico. Para formalizar el concepto definimos:

- $\boldsymbol{o_i}$ $lugar\ geográfico\ origen\ del\ viaje\ \ i$
- $\boldsymbol{d_i}$ $lugar\ geográfico\ destino\ del\ viaje\ i$

Dado un recorrido con $\boldsymbol{N}$ viajes, decimos que dicho recorrido es factible sii:

$\forall\ i\ con\ \ \mathbf{0} < i < N\ \ se\ cumple\ que$

$$\begin{cases} \boldsymbol{d_i} = \boldsymbol{o_{i+1}} \\ \boldsymbol{hf_i} = \boldsymbol{hc_{i+1}} \end{cases}$$

Se puede observar que todos los viajes activos que pertenezcan a un recorrido factible cumplirán la propiedad de ser compatibles entre sí. Se puede pensar en un recorrido factible como una secuencia de viajes activos compatibles ordenados a la que, luego, se le completan los "huecos horarios" con viajes expresos, si es que el hueco se da entre dos lugares geográficos distintos, o viajes esperas, si el hueco se presenta entre distintos horarios en un mismo lugar.

3.1.5 Libro de servicios

El libro de servicios es el conjunto de servicios que se obtienen en la etapa de planificación operativa. Un servicio será realizado por un único vehículo y se define como la secuencia total de viajes, activos, expresos y espera, que dicho vehículo deberá realizar a lo largo de un día. Estos viajes se encuentran a su vez agrupados dentro de cada servicio de manera de representar los distintos turnos que lo componen. La propiedad fundamental de un libro de servicios es que debe hacer un cubrimiento total de los viajes activos definidos en la etapa de planeamiento táctico con la restricción de que cada uno de dichos viajes debe realizarse una única vez.

En resumen los viajes se agrupan en turnos, éstos a su vez se agrupan para formar el servicio de un vehículo y el conjunto de todos los servicios generados a partir de los viajes de entrada forman el libro de servicios.

3.1.6 PET: punto espacio-tiempo

El Punto Espacio-Tiempo (PET) es un concepto importante que surge como resultado de las distintas propuestas de codificaciones posibles planteadas a lo largo de la etapa de análisis del presente proyecto. Decimos que se transformó es nuestra piedra

angular ya que a partir de él se construye todo lo demás. Empezaremos definiendo algunos conceptos básicos para luego aplicarlos en el marco de nuestro proyecto y finalmente a la definición de PET.

En primer lugar es necesario introducir la entidad más simple en un espacio: el punto. Un punto se define como una localización, un lugar, una referencia dimensional. Un espacio de dimensión cero. Dimensión es la característica mínima que nos permite distinguir en un conjunto un punto de otro, la cual nos indica cómo podemos llegar de un punto a otro. Se les llama grados de libertad o dimensiones de un espacio a la característica mínima permitida para todos los puntos del espacio, por eso se dice que el punto no tiene dimensión. Como ejemplo de espacios que sí tienen dimensiones tenemos el plano, el cual es un espacio de dimensión dos, y el volumen, el cual es de dimensión tres.

3.1.6.1 Espacio-Tiempo en Marco Polo

Por definición un mapa es una representación geográfica de la Tierra o de parte de ella en una superficie plana. Relacionando al mapa con los conceptos introducidos anteriormente resulta interesante pensar en él como un objeto de dos dimensiones que se utiliza para representar algunas de las características que existen en la realidad. Existen diferentes tipos de mapas y lo que diferencia a cada uno es el tipo de información que estos extraen de la realidad. La clave, entonces, es abstraer aquello que vamos a precisar y es hacia esa dirección a la que nos dirigimos en esta sección. Para representar un viaje en un mapa podemos usar la información de que éste, por definición, comienza y termina en determinados lugares geográficos. Pero, para tener correctamente determinado al viaje en ese mapa, falta un elemento esencial que es contar con los horarios, tanto de partida desde el lugar geográfico de comienzo, como de arribo al lugar geográfico de destino. Para representar esto, e incorporarlo a los conceptos que venimos manejando, vamos a agregar una nueva dimensión al mapa para representar el tiempo y obtener así un espacio-tiempo de tres dimensiones.

El espacio-tiempo que vamos a representar puede ser imaginado como un volumen de 3 dimensiones (Figura 3.2). El plano formado por los ejes X e Y quedan asociados al mapa, y la recta en el eje Z queda asociada al tiempo.

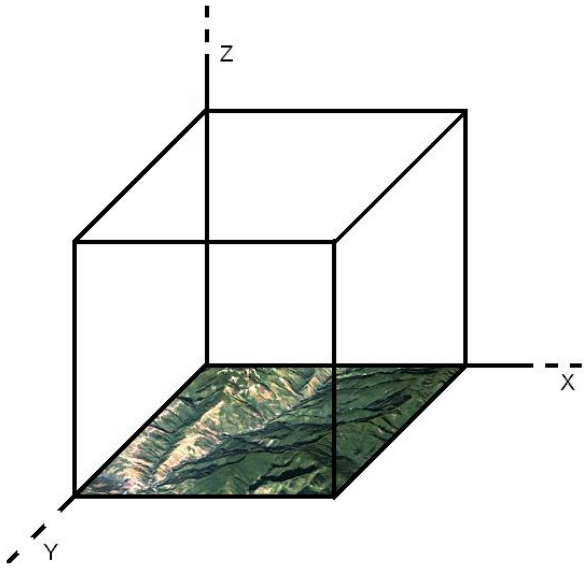

Figura 3.2: Representación del espacio-tiempo

El mapa toma la porción de superficie en la tierra que queremos representar y el tiempo es el de un día, o sea 24 horas, que es el horizonte de planificación considerado. Por lo tanto, un punto dentro de este volumen, se interpreta como el estar situado en un espacio geográfico a una hora determinada del día. Este espacio-tiempo es continuo de modo que movernos en el eje de las X e Y implica un desplazamiento en la posición geográfica mientras que hacerlo en el eje Z implica un desplazamiento en el tiempo.

Finalmente, cuando discretizamos el espacio-tiempo, aparece la definición de PET. En cuanto a la definición se refiere no es estrictamente necesario indicar la manera en que se realiza dicha discretización. Esta puede ser arbitraria, por ejemplo, cuadricular el mapa de a kilómetros o cuadras y el tiempo en horas, o dividir el mapa en manzanas y el tiempo en tres períodos (mañana, tarde y noche). Lo importante en la definición de PET es mencionar que es el subconjunto del espacio que queda delimitado en la discretización. Entonces un PET referencia a un lugar geográfico en un momento de tiempo.

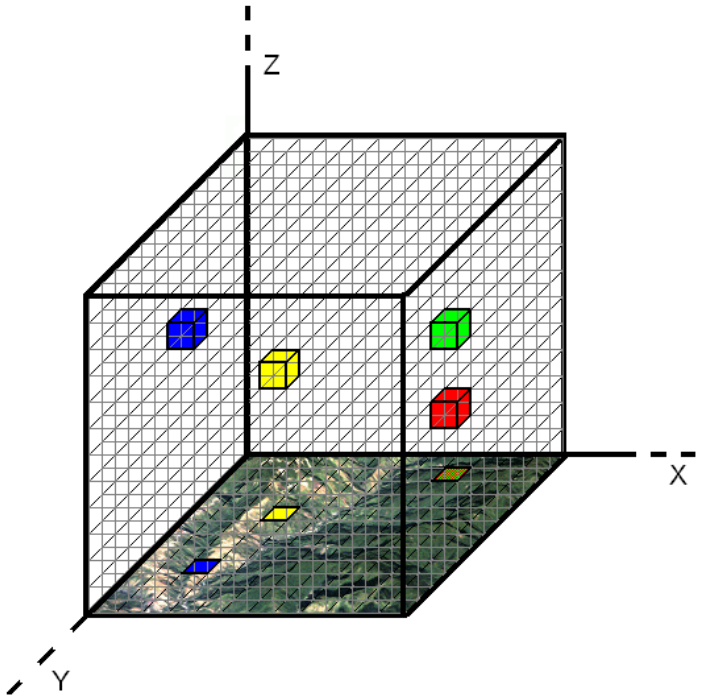

Figura 3.3: Representación de varios PET en el espacio-tiempo

En la Figura 3.3 se presentan cuatro PET diferenciados con distintos colores. Se puede ver como los PET rojo y verde representan distintos horarios en un mismo lugar geográfico, mientras que los PET amarillo y verde representan distintos lugares geográficos en un mismo horario.

En Marco Polo definimos la discretización del tiempo en minutos y la del mapa de manera que en cada partición quede representada una ciudad. De esta manera, un PET representa a una ciudad en un minuto del día. A su vez los PET que nos van a interesar, los cuales serán los únicos que vamos a generar para trabajar, son aquellos que cumplen con al menos una de las siguientes condiciones:

- Existe un viaje activo en donde el lugar geográfico y el horario de **llegada** coinciden con la ciudad y horario del PET

- Existe un viaje activo en donde el lugar geográfico y el horario de **partida** coinciden con la ciudad y horario del PET

Luego de introducidos los PET, el concepto de viaje toma un nuevo sentido. En lugar de decir que un viaje comienza en un lugar geográfico origen a una hora dada y que llega a un lugar geográfico destino a otra hora posterior podemos decir que un viaje queda definido por dos PET, uno de origen (donde se inicia el viaje) y otro de destino (donde se finaliza). A modo de ejemplo, en la Figura 3.4, se presenta una lista con cuatro viajes activos definidos y como se representarían estos viajes en términos de sus PET origen y destino.

Viaje 1: Montevideo -> Pando. Hora salida: 08:00 hs, hora llegada: 09.00 hs
Viaje 2: Pando -> Montevideo. Hora salida: 08:00 hs, hora llegada: 09.00 hs
Viaje 3: Montevideo -> Pando. Hora salida: 09:00 hs, hora llegada: 10.00 hs
Viaje 4: Pando -> Montevideo. Hora salida: 10:00 hs, hora llegada: 11.00 hs

Figura 3.4: Representación de viajes

En primer lugar, en base a los viajes activos definidos en el ejemplo, se crearán los siguientes PET de manera que queden representados todos las parejas ciudad, horario necesarias:

- PET 1: Montevideo-8:00
- PET 2: Pando-8:00
- PET 3: Montevideo-9:00
- PET 4: Pando-9:00
- PET 5: Pando-10:00
- PET 6: Montevideo-11:00

La Figura 3.5 muestra como quedaría la representación gráfica en el espacio-tiempo de los viajes activos del ejemplo:

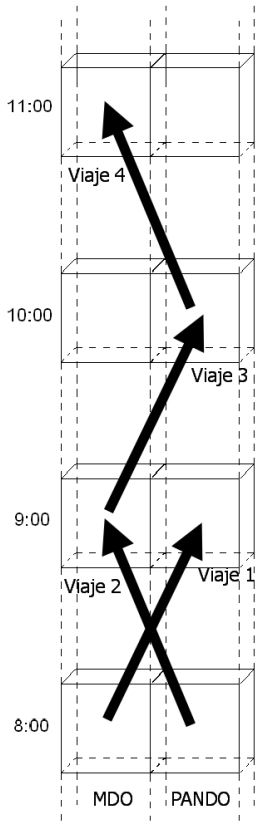

Figura 3.5: Representación de los viajes en el espacio-tiempo

Cabe aclarar que, si la lista de viajes activos definidos en el ejemplo (Figura 3.4) representase la totalidad de viajes activos existentes en la instancia del problema,

entonces los únicos PET que existirían en la solución serían los mencionados anteriormente (PET 1 hasta 6). Luego, a partir de estos PET se construyen todos los viajes expresos y espera posibles, los cuales quedarán disponibles para su uso al momento de armar las soluciones en caso en que se necesiten. Esto puede verse en la Figura 3.6 para el ejemplo que venimos viendo.

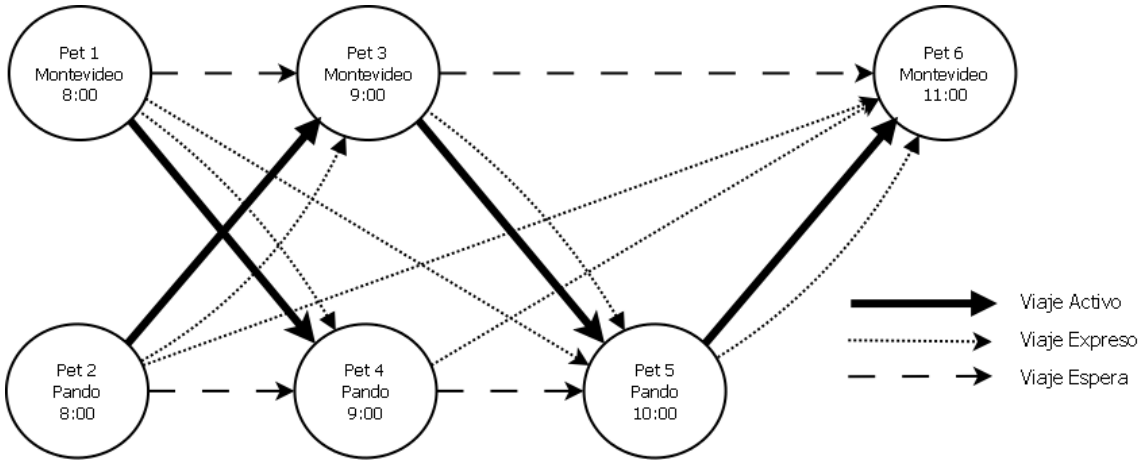

Figura 3.6: Viajes activos, espera y expresos generados

Con respecto a la generación de viajes espera es importante observar que todo PET tendrá siempre un solo viaje espera de salida y un solo viaje espera de entrada. Esto es porque solo se representa el viaje espera que llega al siguiente PET que exista para la misma ciudad y con el horario posterior más cercano al del PET origen. De esta manera una espera más larga deberá representarse con varios viajes espera. En el ejemplo, si quiero representar una espera en Montevideo desde las 8:00 hasta las 11:00 tengo que hacerlo con dos viajes espera consecutivos: uno desde el PET 1 al PET 3 y otro desde el PET 3 al PET 6.

3.2 Conceptos relacionados al método propuesto de solución

3.2.1 Alelo

El alelo fue implementado precisamente para representar la estructura homónima que contiene la información genética de cada gen en un individuo. En cada solución existirán tantos alelos como PET se hayan generado a partir de los viajes activos existentes en el dominio del problema. Cada alelo tiene una relación implícita con su PET con lo cual tiene asociado un lugar geográfico y una hora determinados y será la representación de los acontecimientos que sucedan en dicho lugar a esa hora para una solución en particular. Conceptualmente se puede pensar en el alelo como en una instancia concreta de un PET. En él iniciarán y finalizarán todos los viajes activos que inicien y terminen en el PET al cual representa pero, en cambio, solo contendrá los viajes espera y viajes expresos necesarios para la solución particular a la cual pertenece.

Cada individuo estará formado entonces por una cadena de alelos, de largo igual a la cantidad de PET que existan en el dominio, quienes a su vez tendrán subestructuras dentro, las cuales serán presentadas a continuación.

3.2.2 Record

Cada una de las subestructuras contenidas dentro de un alelo es denominada record. Un record es una estructura implementada para representar la relación de correspondencia entre un viaje de entrada y uno de salida, esto es, entre un viaje que llega al alelo y uno que sale del mismo. Los records representan los ruteos entre los viajes generándose de esta manera cada una de las secuencias de viajes a ser realizadas en una solución. El record puede tener uno de sus viajes, el de entrada o el de salida, nulos para representar el comienzo o el fin de una determinada secuencia de viajes. Cada viaje que llega al alelo en una solución debe tener un record que le asocie un viaje de salida o que indique el final de la secuencia.

Cada alelo tendrá dos conjuntos de records. Uno de ellos para representar el ruteo de cada vehículo, esto es, para modelar cosas como *"el vehículo que llega con el viaje i sale con el viaje j"* o *"el vehículo que llega con el viaje k termina su recorrido en este punto"*. El otro conjunto será para representar, de manera análoga, el ruteo cada turno que llega con cada uno que sale.

3.2.3 Recorrido-AE

Recordemos que un record pertenece a un alelo y un alelo puede contener varios records. En cada alelo se distinguen los records que son asociados a los vehículos y los que son asociados a los turnos.

El recorrido-AE representa la implementación del concepto "recorrido" en un individuo definido en términos del algoritmo genético y se corresponde con una secuencia ordenada de records. El orden en esta secuencia está dado por el horario del PET vinculado al alelo al que pertenece cada record. Dicho de otra manera, un recorrido-AE, se corresponde con una secuencia de viajes que se realizan de manera consecutiva en un individuo.

En cada individuo se distinguen dos tipos de recorridos-AE, los que son realizados por los turnos y los que son realizados por los vehículos. Se parte de la premisa que un turno está asociado a un único vehículo y que un vehículo será conducido por uno o varios turnos, siempre siendo turnos secuenciales, uno a continuación del otro, y siendo generalmente dos o tres turnos en total. En consecuencia tendremos que todos los alelos asociados al recorrido-AE de un turno también estarán asociados al recorrido-AE del vehículo al cual pertenece.

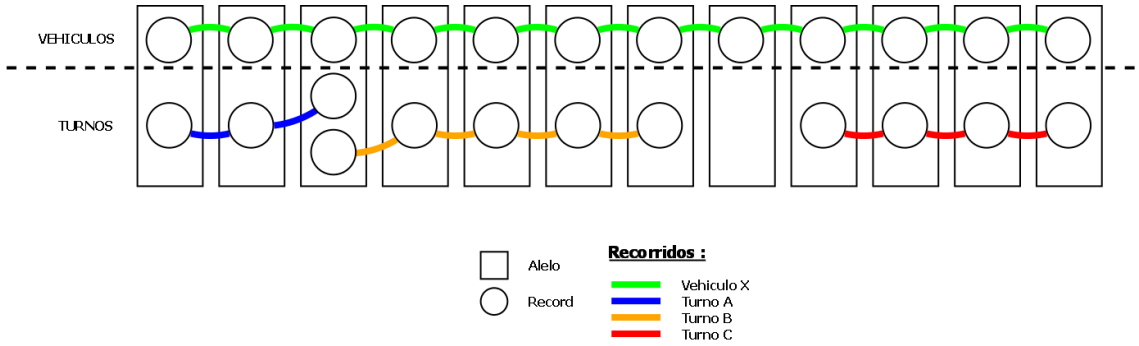

Figura 3.7: Recorridos-AE

En la Figura 3.7 se ve el recorrido-AE del vehículo X el cual tiene 3 turnos asociados. El turno A con 2 viajes, el turno B con 4 viajes y el turno C con 3 viajes. El primer cambio de turno se hace en el mismo alelo mientras que el segundo no. Estas dos alternativas generan escenarios distintos ya que en el primer caso hay dos records asociados al mismo alelo, mientras que en el segundo hay un alelo que no tiene records de turnos. Cuando un turno termina y el siguiente no empieza en el mismo alelo, el recorrido-AE del vehículo paralelamente realizará viajes esperas para que de este manera el vehículo se encuentre disponible en la misma ciudad para el siguiente turno. En caso de tratarse de un vehículo con un solo turno, el recorrido-AE del vehículo será igual al recorrido-AE del turno.

Capítulo 4: Estudio Analítico

Debido a que la complejidad del problema al que nos enfrentamos es NP-difícil (15), el enfoque adoptado para resolver la situación planteada radica en la utilización de una meta-heurística. En particular la solución propuesta consiste en la elaboración de un algoritmo genético. El presente proyecto de grado se presenta como sucesor del PG42 (2) el cual tenía por principal objetivo la creación de una solución que permita la obtención de libros de servicio buscando minimizar costos operativos, considerando una realidad modelada y definida a través de parámetros, restricciones y reglas. En busca de conseguir lograr los objetivos trazados para el presente proyecto, la base del desarrollo se establece a partir de dos grandes tareas iniciales las cuales se desarrollan en este capítulo.

La primer tarea consta del estudio y análisis del PG42 de manera de asimilar el problema y sus antecedentes bibliográficos y decidir si la parte de implementación de dicho proyecto se utiliza como base del actual, si se reutilizan módulos y/o ideas, o si, por el contrario, se plantea la solución desde una nueva perspectiva. En el análisis se presenta un resumen bibliográfico del PG42 para contextualizar nuestro trabajo y se analiza el diseño de las distintas estructuras de datos y los diferentes módulos implementados con visión crítica de manera de poder decidir si continuar el desarrollo existente o diseñar nuevamente alguna o todas las estructuras utilizadas.

La segunda tarea consiste en el diseño y la implementación de los conceptos, operadores, módulos y estructuras auxiliares necesarias para instanciar el problema que luego se resolverá con un algoritmo genético haciendo uso de un motor de resolución que provee los esqueletos algorítmicos necesarios. Ambas tareas están fuertemente ligadas en el sentido de que los resultados obtenidos en la primera definen enteramente la forma de encarar la segunda.

4.1 Análisis

4.1.1 Proyecto de grado del año 2009

En el PG42 se introduce el problema de ordenamiento de flota y personal estudiando los distintos trabajos existentes en la bibliografía que lo han abordado. Se menciona también como los trabajos más recientes, que lo resolvieron desde un enfoque integrado, arrojaron mejores resultados que los que lo hicieron usando enfoques secuenciales.

Dentro de la tarea de ordenamiento de flota y personal en los problemas relativos al transporte público (1) se pueden distinguir dos grandes sub-tareas de asignación, las cuales son, la asignación de los vehículos de la flota y la asignación de la tripulación. Por un lado se debe resolver la asignación de los vehículos a los viajes, problema que se conoce por la sigla VSP por “Vehicle Scheduling Problem”. Los objetivos del VSP son minimizar el tamaño de la flota requerida y evitar o minimizar los kilómetros expresos realizados por los vehículos. Por otro lado se debe resolver también el problema que asigna la tripulación a dichos viajes, problema denominado DSP, “Duty Scheduling Problem”. Los objetivos del DSP son minimizar la cantidad total de jornales y, dentro

de estos, minimizar la cantidad de horas de trabajo con la restricción de que se contemplen las normas laborales vigentes. Estos dos problemas fueron encarados en principio de forma separada y secuencial (5). En el modo secuencial se suele resolver en primer lugar el VSP y luego, sobre esta asignación ya resuelta, el DSP y de esta manera se llega a la resolución de ambas tareas.

A finales de los noventa aparecen los primeros enfoques para resolver los problemas VSP y DSP de manera integrada. Dicho problema es denominado IDVSP, sigla de "Integrated Duty Vehicle Scheduling Problem". El IDVSP ataca simultáneamente los dos problemas de asignación resolviendo la asignación de la flota de vehículos y generando a su vez los jornales de trabajo que deben ser cumplidos por la tripulación. Los objetivos que persigue el IDVSP son, consecuentemente, la unión de los objetivos perseguidos por los problemas VSP y DSP.

El PG42 se plantea resolver el problema IDVSP utilizando como estrategia de resolución un Algoritmo Genético. Dado que el Marco Polo se plantea como proyecto sucesor del PG42, y persigue resolver el mismo problema con la misma estrategia de resolución, a continuación se presenta un análisis, de las distintas estructuras y técnicas definidas y utilizadas en el proyecto predecesor, a partir del cual se determina el rumbo a seguir.

4.1.1.1 Representación de la solución

En el PG42 se define la codificación de la solución buscando conseguir una representación simple, que permita rápido acceso a la información que esta contiene y bajo costo de almacenamiento. Bajo esta premisa se codifica la solución con dos vectores que representan, el primero, la asignación de la flota a los viajes activos y, el segundo, la asignación de turnos. Ambos vectores tienen el mismo largo el cual es igual a la cantidad de viajes activos del problema. El índice en cada vector representa el identificador del viaje activo en cuestión y el valor contenido dentro de la celda para cada índice indica el identificador del vehículo que lo realizará, en el caso del primer vector, y el identificador del turno en el caso del segundo vector. Esto es, en el vector *vehiculos*, *vehiculos[i]* contiene el identificador del vehículo que realiza el viaje activo i. Funciona de manera análoga para el caso de los turnos. El identificador de turno corresponde a un chofer para el cual luego se generará el detalle total del turno en forma de eventos a realizar, esto es, viajes activos, expresos, descansos, etc.

T1	V1	A
T2	V2	C
T3	V3	A
T4	V4	E
T5	V5	E
T6	V6	D
T7	V7	B

Figura 4.1: Representación de la solución en PG42

La simplicidad de esta representación parece indiscutible y se destaca que la propiedad de cubrimiento total de los viajes activos se cumple de antemano por la forma en la que está definida. Pero por otro lado, dada la semántica de los datos almacenados en esta estructura tan simple, el armado del libro de servicios resulta en una recorrida exhaustiva de toda la estructura. Esto no sería un problema si el armado del libro se hiciera solo una vez, por ejemplo, al terminar la ejecución del algoritmo. En la práctica esta estructura se recorre, para armar el libro de servicios que representa, cada vez que una solución es sometida a un operador evolutivo para chequear la factibilidad de la misma y también en cualquier caso en el que se necesite calcular su costo o valor de aptitud. El armado es necesario dado que los descansos y los viajes expresos no están explícitamente representados en la estructura y solo se logran visualizar una vez que se arma, para cada vehículo y para cada turno, la lista de viajes activos asignados. Esta característica termina comprometiendo la simplicidad que se visualiza a primera vista.

Con respecto a la resolución de las dos tareas de asignación sobre esta estructura es cierto que estas se resuelven de manera integrada si uno mira el algoritmo como un todo, es decir: al término de cada iteración las soluciones incluidas en la población tienen ambas asignaciones resueltas en la estructura al igual que en el momento en que el algoritmo finaliza y arroja el mejor resultado. Pero si miramos el tratamiento específico que se le hace a una solución al momento de operar sobre ella vemos que en realidad en el algoritmo genético se manipula solamente el vector de vehículos resolviendo luego con técnicas constructivas la asignación de turnos. Si se analiza desde esta última perspectiva, podemos decir que internamente se termina haciendo un tratamiento secuencial de ambas tareas de asignación.

4.1.1.2 Técnicas de inicialización

En el PG42 se definen y desarrollan cuatro técnicas para generar soluciones iniciales:

- Inicialización Secuencial: la idea es asignar a un mismo vehículo viajes compatibles secuenciales de manera de minimizar el kilometraje expreso que se generaría si el próximo viaje activo asignado comenzara en un lugar diferente al destino del anterior viaje activo.

- Inicialización Aleatoria: la idea es aportar diversidad a la población armando soluciones asignando viajes a vehículos y turnos de manera aleatoria, siempre que esta asignación resulte factible.

- Inicialización por Entronques: este método agrupa los viajes de ida y vuelta entre dos puntos e intenta asignarlos al mismo vehículo. La idea está inspirada en una práctica común de las empresas de transporte.

- Inicialización combinada: esta técnica combina las tres técnicas anteriores seleccionando, cada vez que se va a generar un individuo, una de dichas técnicas en base a un valor probabilístico dado.

Todas las técnicas tienen fundamentos lógicos y fueron definidas bajo la premisa de generar individuos que sean factibles y que cubran todos los viajes activos. Generan soluciones iniciales con técnicas constructivas.

4.1.1.3 Función de fitness

La función de costos definida en el PG42 intenta considerar los distintos objetivos perseguidos por el IDVSP. Siguiendo la bibliografía plantean la utilización de una función multiobjetivo que contemple los distintos intereses o funciones objetivo combinándolas linealmente. De esta manera se consigue que el problema implique alcanzar un único objetivo, la minimización de la combinación lineal definida. Plantean minimizar las siguientes cantidades: kilómetros expresos, vehículos utilizados, jornales y horas totales trabajadas y además agregan una penalización por turnos que no alcancen una duración mínima. Se definen funciones que calculan cada una de estas cantidades y luego se combinan linealmente utilizando coeficientes para escalarlas y definir la importancia o peso de cada una de ellas. Estos coeficientes se calibran previamente y se toman luego como entrada de la ejecución del algoritmo.

4.1.1.4 Operadores genéticos

Los operadores de cruzamiento y mutación, al operar directamente con la información genética contenida en la solución, suelen estar muy atados a la forma en que la misma se codifica. Esto se cumple en el PG42 en donde se implementan las técnicas de cruzamiento en un punto y una mutación en la cual se sortean un conjunto de posiciones del vector solución y se cambian sus vehículos asignados seleccionando aleatoriamente vehículos de entre los que se encuentren disponibles.

En cuanto al operador de selección, la implementación de distintos métodos de selección viene incluida en el motor de esqueletos algorítmicos. Este operador se presenta en el PG42 como parte de la calibración para determinar, entre el operador de Rueda de Ruleta y el de Selección Estocástica Universal, cual es el que mejor se comporta en el escenario planteado. En el PG42 (2) se afirma que, si bien no se observaron grandes diferencias entre ambos operadores en las pruebas realizadas, se nota una pequeña mejora en los costos que se obtienen con selección estocástica pero se empeora sensiblemente el tiempo de ejecución. De todos modos se optó por utilizar este operador.

4.1.1.5 Módulo de reglas laborales

En el PG42 se desarrolla un módulo (WRM) que gestiona las funcionalidades referentes a las reglas laborales. Motivados por las necesidades de actualización permanente de dichas reglas dada la realidad cambiante y de flexibilidad en su especificación, definen e implementan un lenguaje orientado a que el usuario sea capaz de definir sus propias reglas las cuales pueden establecer restricciones sobre cualquiera de los conceptos modelados. Las reglas deben ser definidas en archivos xml que luego son interpretados por el módulo con la asistencia de una biblioteca externa. Este módulo es el encargado de determinar la factibilidad de las soluciones en lo que respecta al cumplimiento de las reglas laborales y de descanso definidas.

4.1.1.6 Conclusión

La tarea de estudiar las estructuras y las distintas técnicas utilizadas en el PG42 junto con los resultados obtenidos nos motiva a plantearnos alternativas que puedan llegar a

mejorar los distintos aspectos en los cuales el proyecto predecesor mostró debilidades. Lo realmente difícil es lograr detectar acertadamente cual es el aspecto que es necesario cambiar o modificar para lograr modificar el comportamiento o resultado que consideramos mejorable.

Decidimos sacrificar la simplicidad de la representación para intentar obtener una estructura que, a cambio de ser más compleja, represente mejor la realidad y en la cual estén representados todos los conceptos necesarios. De todas maneras, de acuerdo al análisis realizado, consideramos que esta simplicidad no es tal, siendo que el costo de manipular una solución es muy elevado, al tener que hacer recorridas exhaustivas para armar y, recién ahí, poder interpretar los datos que esta contiene. Recordemos que ni los viajes expresos ni los viajes espera están representados de forma explícita en la estructura del PG42, sino que estos deben calcularse cada vez deduciendo en cada caso que acción se debe realizar entre dos viajes activos consecutivos. El impacto de esta decisión de diseño se refleja en la complejidad encontrada en el código del PG42, sobre todo en las funciones de armado de turnos y control de la factibilidad de la solución. Por otro lado no logramos visualizar de qué manera se podría resolver el problema integrado con una estructura de estas características, en el sentido de hacer que los operadores genéticos operen sobre los datos de ambos vectores, sin hacer recorridas completas sobre ambas estructuras. Esto nos motiva aún más a decidir invertir tiempo en diseñar una nueva codificación que se adapte mejor a nuestras necesidades.

El hecho de decidir no utilizar la misma forma de codificar la solución anula la posibilidad de reutilizar cualquier otro módulo cuya implementación sea dependiente de dicha codificación. Esto no significa que no se puedan tomar las ideas que se consideren útiles, sino que si se quieren reutilizar éstas tienen que ser implementadas nuevamente.

Adoptamos las técnicas de inicialización de soluciones planteadas en el PG42 las cuales deben ser re-implementadas para generar soluciones iniciales adecuadas a nuestra codificación. Por otro lado tomando como lección aprendida la importancia de la calidad de dichas soluciones y la cantidad de tiempo que consume esta generación decidimos aislar esta tarea en un módulo especialmente implementado para gestar soluciones iniciales (Anexo D).

Luego de analizar los resultados obtenidos en el PG42 en el proyecto Marco Polo se deciden dos cuestiones básicas en lo que respecta a la función de fitness. Una de ellas es para atacar el problema de la evaluación de múltiples funciones con codominios potencialmente muy distintos para lo cual se decide transformar todas las cantidades involucradas a dinero. Esto se parametriza mediante un archivo de configuración en el cual se indican los distintos costos en dinero de cada uno de los conceptos manejados, como ser, el valor de la hora de trabajo y el precio del combustible. De esta manera se ponderan naturalmente los distintos conceptos incluidos en el cálculo y se terminan usando, como valor de importancia relativa, directamente los costos asociados a los distintos rubros. En definitiva se adoptó la misma decisión tomada en el PG42 en cuanto a modelar la función de fitness como una combinación lineal de los distintos objetivos, solo que dejamos que las funciones combinadas se ponderen de manera

natural. Esto trae como beneficio contar con una tarea de calibración menos, lo cual no es un detalle menor, gracias a que las variables que eran posibles candidatas a calibrar quedan definidas por datos reales del escenario donde es aplicado el algoritmo. La otra decisión tomada con respecto a las funciones que integran la función de fitness es no tomar en cuenta el objetivo de minimizar la cantidad de vehículos a utilizar. Consideramos que tener inoperativas las unidades que componen la flota también incurre en un costo para la empresa el cual no es tan sencillo de calcular por lo que decidimos centrarnos en la minimización del costo final real.

En cuanto a los operadores de cruzamiento y mutación encontramos que no solo la implementación es dependiente de la codificación sino que también la idea detrás de los mismos se basa en dicha codificación por lo que consideramos que no se pueden reutilizar. El operador de selección utilizado en el PG42 es el operador conocido como Selección Estocástica Universal, luego de haber realizado pruebas también con el operador de Selección por Rueda de Ruleta concluyendo que no se encontraron grandes diferencias entre ambos. En el presente proyecto, en cambio, se opta por utilizar el operador de Selección por Rueda de Ruleta debido a que, en este, se define que los individuos más aptos tienen una probabilidad más alta de ser seleccionados, pero por otro lado, preserva la diversidad en la población permitiendo que hasta los peores individuos puedan ser elegidos. Además, este método de selección, tiene la ventaja de que no agrega ningún parámetro el cual se deba calibrar.

Heredamos la idea de aislar también la gestión y validación de las reglas laborales en un módulo independiente no pudiendo tampoco en este caso reutilizar el WRM por estar fuertemente atado a la codificación de su solución. Nos proponemos lograr independizar nuestro módulo de la codificación elegida definiendo interfaces que, en caso de ser respetadas, permitan la comunicación con dicho módulo más allá de cómo esté representada la solución. Por otro lado consideramos que el lenguaje definido en el PG42 se queda en un término medio entre un lenguaje orientado a un usuario final común y uno con perfil programador. La simpleza en cuanto a las directivas definidas parecen indicar que un usuario común podría definir sin problemas las reglas pero, por otro lado, los conceptos referenciados por dichas reglas deben ser implementados por un usuario con conocimientos en programación. Encontramos que lo más razonable es orientar la posibilidad de extensión de las reglas laborales enteramente a un usuario programador y creemos que lo adecuado para un usuario final común sería contar con una interfaz gráfica adecuada para definir reglas lo cual escapa al alcance del presente proyecto.

Capítulo 5: Algoritmo genético

Para construir un algoritmo genético es necesario partir de una realidad, identificar las soluciones y transformarlas en una representación de modo tal que el algoritmo genético pueda computar sobre las mismas. A este proceso se le denomina codificación de la solución. Una vez establecida la codificación de la solución es necesario determinar los operadores evolutivos, llámense selección, cruzamiento y mutación. En adición debe especificarse la función de aptitud, la cual indicará cuales soluciones son mejores o más aptas que otras. Otro aspecto a considerar son las restricciones que deben ser aplicadas a las soluciones para determinar la factibilidad de las mismas. Finalmente y no siendo menos importante es necesario definir como las soluciones iniciales serán generadas.

5.1 Codificación de la solución

El principal objetivo a perseguir al momento de elegir la representación de la solución para el problema dado, radica en almacenar en los genes toda la información necesaria para conseguir resolver el problema a través de la propia evolución, sin necesidad de introducir mecanismos deterministas o aleatorios ajenos a los propios del algoritmo genético.

Para lograr codificar una solución en primer lugar es necesario definir qué es lo que se espera tener como solución al problema. En este caso, la solución al problema planteado, consiste en la generación de un libro de servicios, cada uno de los cuales se compone de turnos que a su vez están compuestos por una secuencia de viajes. Se hace notar que, por definición de libro de servicios, en la solución deben estar presentes todos los viajes activos del problema. La codificación elegida se basa en los conceptos definidos anteriormente: los alelos, los cuales representan instancias concretas de los PET existentes en el problema. El cromosoma que representa a la solución, consiste en un vector de Alelos cuyo largo será igual a la cantidad de PET definidos en el problema.

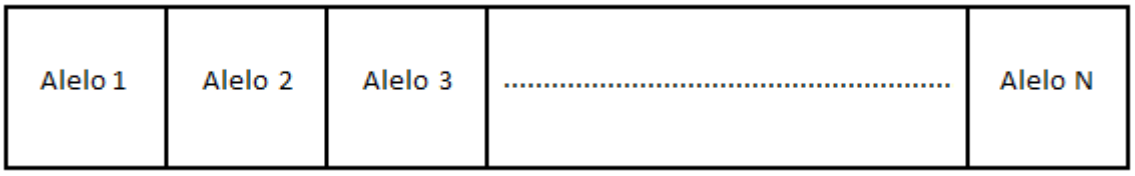

Figura 5.1: Vector de Alelos

El PET, como lo indica la sigla que forma su nombre, es un punto en el Espacio-Tiempo. Para estos puntos bidimensionales se cuenta, como dato de entrada, con el conocimiento de todos los viajes que arriban y todos los que parten de cada uno de ellos.

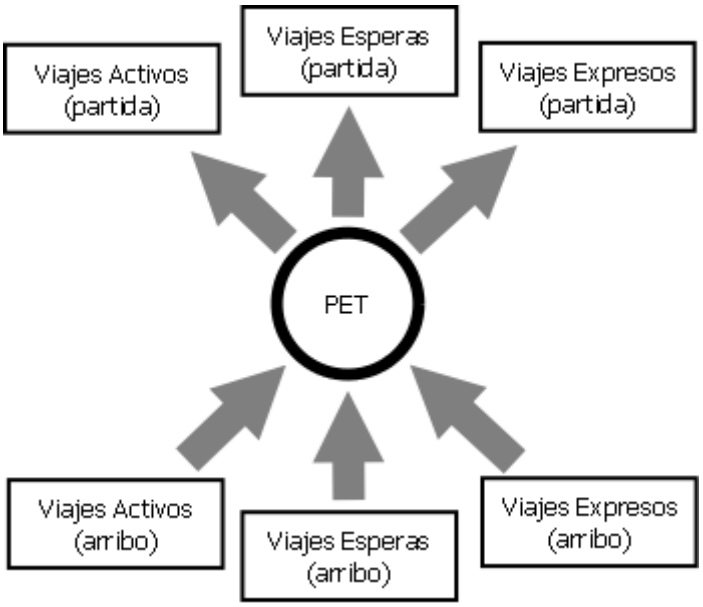

Figura 5.2: Viajes que parten y llegan a un PET

Luego, en el alelo, obligatoriamente existirán las entradas y las salidas que se correspondan a viajes activos y opcionalmente existirán los viajes espera y expresos que sean requeridos según como esté armado el individuo. En base a este conocimiento se almacenan en el alelo relaciones de correspondencia, uno a uno, entre los viajes que arriban y los viajes que parten. A su vez, para cada una de estas correspondencias se almacenan dos tipos de relaciones: una referente a los vehículos y la otra a los turnos. Estas relaciones se modelan con vectores de records contenidos dentro de cada alelo.

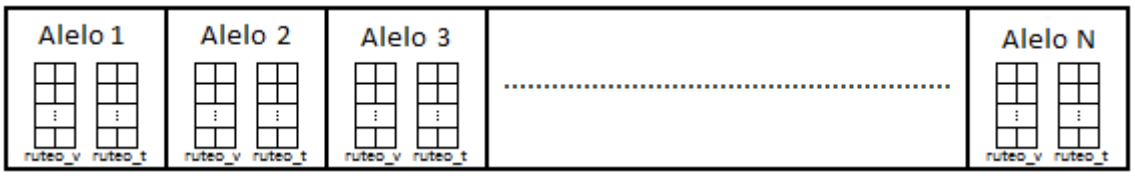

Figura 5.3: Vector de Alelos con vectores de Record

Estos datos constituyen la información genética contenida en cada uno de los alelos y representan los ruteos entre los viajes que arriban y los viajes que parten de cada uno de los alelos. El objetivo del algoritmo genético radica en encontrar los ruteos que optimicen los objetivos planteados.

En la Figura 5.4 se puede ver un ejemplo de viajes que arriban y parten de un alelo i y los ruteos de vehículos y turnos representados con colores verde y rojo respectivamente. La información que debe ser interpretada a partir del diagrama es la siguiente:

- El vehículo que arriba al alelo i con el viaje arribo 1 parte con el viaje partida 1
- El vehículo que arriba al alelo i con el viaje arribo 2 parte con el viaje partida 2
- El turno que arriba al alelo i con el viaje arribo 1 parte con el viaje partida 2
- El turno que arriba al alelo i con el viaje arribo 2 parte con el viaje partida 1

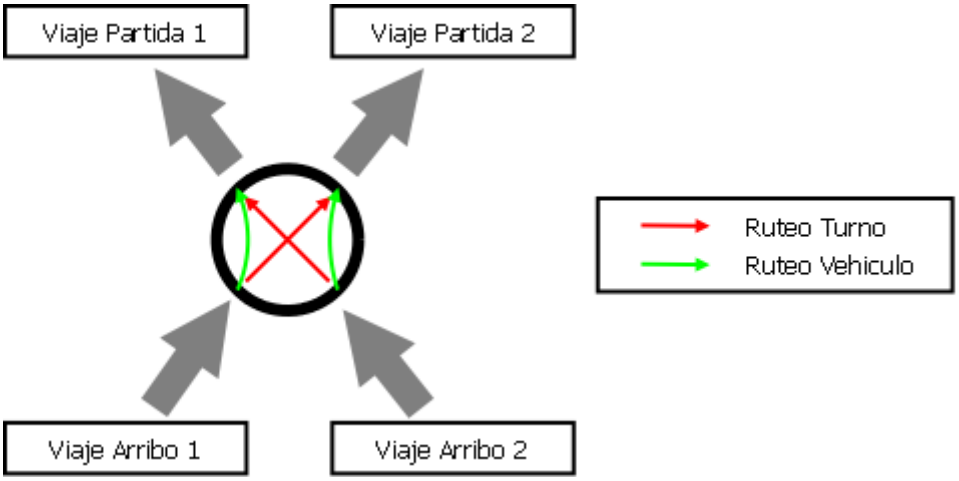

Figura 5.4: Información contenida en el Alelo

Siempre que en la solución exista un viaje, activo o expreso, que parta de un alelo debe existir una relación de turno y una de vehículo asociada a dicho viaje de manera que este viaje tenga asignados vehículo y tripulación para poder llevarse a cabo. Cabe destacar que, en el caso de que se trate de un viaje espera, solo debe existir la relación de turno ya que el vehículo se mantiene en un mismo lugar a lo largo del tiempo. Esto, sumado al motivo por el cual se genera un PET en la solución (3.1.6), asegura que todos los viajes activos serán cubiertos tanto por un turno como por un vehículo en cada individuo. A su vez es importante mencionar que el archivo de entrada del problema, el cual contiene los viajes activos a cubrir generados en la etapa de planeamiento táctico, determina todos y cada uno de los PET que existirán en toda solución generada en base a esta codificación. Esta determinación viene dada por los lugares y horarios de comienzo y fin de todos los viajes activos existentes. Luego, cada solución contendrá distinta información genética dentro de cada alelo y podrá contener además viajes expresos y viajes espera pero solo si estos comienzan o finalizan en PET que hayan sido determinados previamente por algún viaje activo.

5.2 Función de fitness

La función de *fitness* aplica sobre una solución particular y el resultado de dicha evaluación determinará cuan buena o apta, si se utiliza la jerga de los algoritmos genéticos, resulta ser la misma. "*Su influencia en el mecanismo del algoritmo es importante ya que pese a que actúa como una caja negra para el proceso evolutivo, la función de fitness guía el mecanismo de exploración, al actuar representando al entorno que evalúa la aptitud de un individuo solución para la resolución del problema*" (2)

Para determinar la aptitud de una solución se tomaran en cuenta principalmente las siguientes variables:

- Kilómetros expresos
- Jornales

Los kilómetros expresos son aquellos kilómetros que un vehículo realiza sin obtener ganancia alguna, pero que sí presentan un costo para la compañía de transporte. En otras palabras, es la distancia recorrida en un viaje expreso. Lo que se espera de este factor es que tienda a cero con el fin de no generar costos que no incluyan ganancias. Así como los kilómetros expresos se encuentran directamente relacionados a los vehículos también tenemos que los jornales están directamente relacionados a los turnos. Un turno tendrá cierta duración en función del trabajo que se le ha sido asignado. En particular lo que se busca de los turnos es que la duración de los mismos sea próxima a la duración del jornal típico definido por la normativa vigente.

Debido a que la función de fitness queda definida en base a más de un objetivo, la aptitud de un individuo quedará determinada con la combinación de los resultados obtenidos a partir de evaluar ambos objetivos sobre dicho individuo. Para representar matemáticamente estos conceptos se define como sigue la función de costos

$$\boldsymbol{c : Solución} \rightarrow \mathbb{R}$$

$$\boldsymbol{c(x)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{i=m}} \boldsymbol{CE}\ (\boldsymbol{i}, \boldsymbol{x}) \ + \sum_{\boldsymbol{i=1}}^{\boldsymbol{i=n}} \boldsymbol{CT}\ (\boldsymbol{i}, \boldsymbol{x})$$

Donde

- $\boldsymbol{CE}\ (\boldsymbol{i}, \boldsymbol{x})$ *función que evalúa el costo del viaje expreso i en la solución x*
- $\boldsymbol{CT}\ (\boldsymbol{i}, \boldsymbol{x})$ *función que evalúa el costo del turno i en la solución x*
- $\boldsymbol{m}$ *es la cantidad de viajes expresos que hay en la solución x*
- $\boldsymbol{n}$ *es la cantidad de turnos que hay en la solución x*

Las funciones que evalúan los costos, tanto de los viajes expresos como de los turnos, expresan su resultado en dinero, tomando en cuenta los costos reales de los insumos y los recursos, de manera de simular la situación real de cálculo de costos de la empresa y así no tener necesidad de utilizar ponderadores para destacar uno de los costos sobre el otro. Esto además agrega la ventaja de no tener que calibrar dichos ponderadores.

La función que calcula el costo de los turnos considera que, en el caso en que el turno dure más de la duración indicada como ideal (valor parametrizable fijado en 8 horas), el excedente de horario debe ser computado como horas extras, incrementando su valor de acuerdo a un parámetro previamente cargado con el factor por el cual debe multiplicarse el costo de la hora de trabajo común para obtener el costo de la hora extra. Análogamente, los turnos cuya duración sea menor a la ideal, tendrán asociado igualmente el costo de un jornal completo sea cual sea su duración real. De esta manera queda implementada una penalización implícita tanto para los turnos de corta duración como para los que tengan una duración excesiva.

Si bien existe la posibilidad de que no hayan viajes expresos en una solución siempre existirá al menos un turno, con lo cual se tiene que:

$$\boldsymbol{c(x)} > 0, \quad \forall\, x \ \boldsymbol{Solución}$$

Hechas estas observaciones y en vista de que el algoritmo genético considera como individuos más aptos aquellos que poseen un valor de aptitud más elevado, la función de fitness se define de la siguiente manera:

$$\boldsymbol{f} : \boldsymbol{Solución} \rightarrow \mathbb{R}, \qquad \boldsymbol{f(x)} = \frac{\mathbf{1}}{\boldsymbol{c(x)}}$$

5.3 Restricciones

Las siguientes son las restricciones que deberá cumplir una solución para ser considerada factible:

1. Cubrimiento total de los viajes activos
2. Los turnos que componen la solución deben ser turnos factibles (Sección 3.1.2.3), esto es, que deben respetar las reglas laborales definidas como válidas en el problema
3. Los recorridos que componen la solución deben ser recorridos factibles (Sección 3.1.4.1)

5.4 Operadores Genéticos

5.4.1 Selección

Luego de analizar las características de los distintos métodos de selección decidimos hacer uso de la técnica de selección proporcional o por rueda de ruleta. Esta técnica determina la cantidad de copias de individuos a seleccionar de manera proporcional a sus valores de fitness.

Sea N el número total de individuos existentes, $\boldsymbol{f}$ la función de fitness del algoritmo y $\boldsymbol{x_i}$ el i-ésimo individuo de la población, la probabilidad que tiene el individuo $\boldsymbol{x_i}$ de ser seleccionado en la selección de rueda de ruleta es la siguiente:

$$\boldsymbol{p_i} = \frac{\boldsymbol{f(x_i)}}{\sum_{\boldsymbol{j=1}}^{\boldsymbol{j=N}} \boldsymbol{f(x_j)}}$$

Esta técnica elige de manera aleatoria individuos dentro de una ruleta sesgada en la cual no todos los individuos tienen igual chance de ser seleccionados sino que sus posibilidades son proporcionales a su valor de fitness. Esto logra que, por un lado, individuos más aptos tengan una probabilidad más alta de ser seleccionados perpetuando así las buenas características que estos poseen pero también, por otro lado, se conserve la diversidad en la población permitiendo que hasta los peores

individuos puedan ser seleccionados. Este comportamiento parece ideal para este tipo de problemas dada la dificultad de lograr individuos con buenas características. Además este método de selección tiene la ventaja de que no agrega ningún parámetro para el cual se deba calibrar su valor como sí lo agregan, por ejemplo, el método de selección por torneo, en el cual se debe elegir el tamaño de la muestra a seleccionar, o el método de ranking, en el cual se debe determinar el porcentaje de la población que usa el método para operar.

5.4.2 Cruzamiento

El principal objetivo a tener en cuenta al momento de definir este operador radica en poder generar nuevos individuos los cuales tienen como principal característica que absolutamente toda su información genética es heredada de sus progenitores. Debido a que la representación de la solución definida no se ajusta a las representaciones comúnmente usadas en trabajos con algoritmos genéticos, los operadores de cruzamiento implementados son operadores específicos definidos por el equipo de trabajo de Marco Polo.

5.4.2.1 Cruzamiento en un alelo

Al ser un individuo un vector o lista de alelos, la idea de este operador de cruzamiento radica en intercambiar la información genética alelo a alelo entre dos progenitores. Antes de aplicar el operador de cruzamiento sobre los individuos seleccionados se debe definir cuál será el alelo que intercambiará su información genética de manera que, en primer lugar, se sortea uno de los alelos existentes en el individuo. Una vez seleccionado el alelo a cruzar hay que hacer una comparación entre la información que contienen ambas soluciones en dicho alelo de manera de determinar el conjunto de ruteos que van a participar activamente del cruzamiento. Se decide que, en este operador, los ruteos que participarán del cruzamiento serán solamente los que establezcan una correspondencia entre viajes que pertenezcan a ambas soluciones. En este conjunto entrarán entonces los ruteos que involucren viajes activos, puesto que por definición las soluciones hacen un cubrimiento total de los viajes activos, y los que involucren viajes expresos y espera que existan en ambas soluciones a la vez. Los ruteos no pertenecientes a este conjunto no presentarán modificación genética alguna por motivos de factibilidad.

Funcionamiento del algoritmo:

El algoritmo implementado para este operador se encargará de realizar el intercambio de las relaciones que existen dentro del alelo a cruzar de los progenitores para el conjunto de ruteos que participará activamente en el cruzamiento. La idea es que la información genética del alelo seleccionado se modifique de manera que a cada viaje de entrada de la primera solución se le asigne el ruteo con el viaje de salida que dicho viaje tiene en la segunda solución y viceversa. Este cambio se hace tanto en términos de los ruteos de vehículos como en los ruteos de turnos.

Dados dos individuos progenitores 1 y 2, siendo el $alelo_i$ el alelo seleccionado para participar del cruzamiento, R_{i1} el conjunto de relaciones que tiene el $alelo_i$ en el primer

progenitor y R_{i2} el conjunto de relaciones que tiene el $alelo_i$ en el segundo progenitor, el resultado de aplicar el operador de cruzamiento es el intercambio del conjunto de relaciones del $alelo_i$ entre los progenitores. En la siguiente ilustración (Figura 5.5) se puede observar como el individuo descendiente 1 posee toda la información genética del individuo progenitor 1 salvo en el $alelo_i$ en donde se encuentra la información genética del individuo progenitor 2. Análogamente el descendiente 2 posee toda la información genética del individuo progenitor 2 salvo por las relaciones contenidas en el $alelo_i$

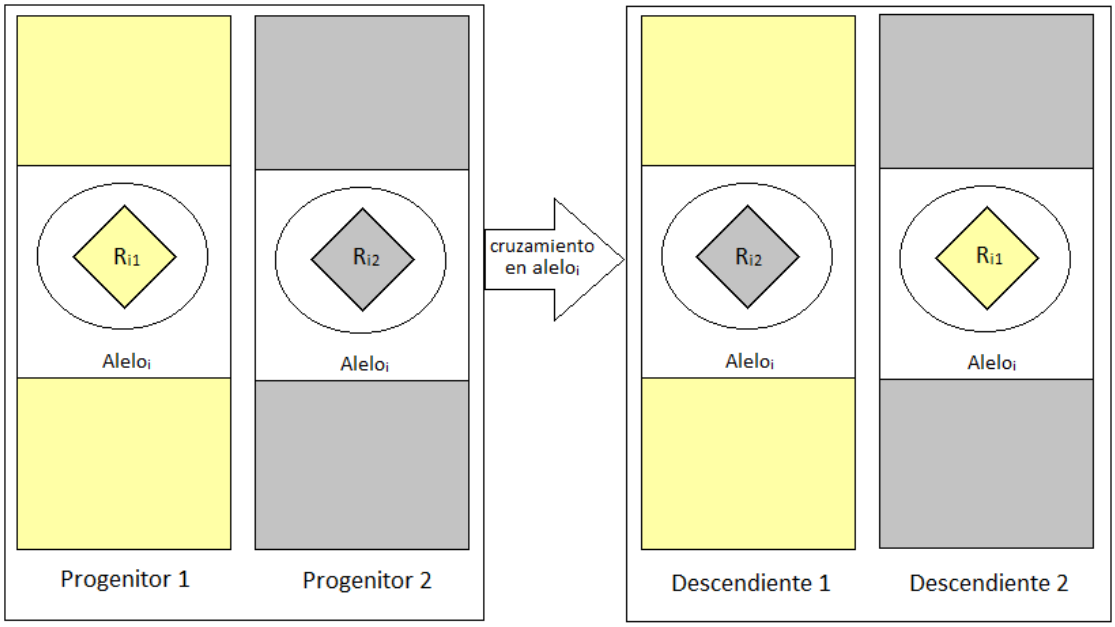

Figura 5.5: Cruzamiento en un alelo

Este intercambio de información solamente se hace efectivo si todas las restricciones impuestas al problema son satisfechas. En caso de que esto no ocurra, el alelo que iba a ser sometido al cruzamiento, permanecerá sin cambios.

Es importante destacar que, dada la codificación de solución definida, este operador genético resulta ser sumamente eficiente desde el punto de vista computacional por la forma en la que está implementada la estructura de datos.

5.4.2.2 Cruzamiento Recursivo de Subgrafos

Este cruzamiento, al igual que el anterior, intercambia la información genética alelo a alelo entre dos progenitores con la diferencia de que, en este caso, se intercambiarán todos los ruteos existentes y se aplicará luego recursivamente el cruzamiento a los alelos afectados por nuevos ruteos o por los ruteos que deban eliminarse. Es importante mencionar que el cruzamiento en un alelo es un caso particular del cruzamiento recursivo de subgrafos. En este caso, el conjunto de ruteos a intercambiar coincide con el conjunto total de ruteos del alelo sorteado y luego de cruzar dicho conjunto, ningún alelo resulta afectado y no hay recursión, sino que se aplica solamente el paso base.

Para definir el operador de cruzamiento recursivo es necesario previamente definir dos conceptos adicionales a los ya considerados los cuales se manejan en el presente algoritmo:

- *viaje espacial*: denominamos viaje espacial a aquel viaje que se mueve tanto en el espacio como en el tiempo. Específicamente en el contexto en el cual nos encontramos un viaje espacial puede ser o bien un viaje activo o bien un viaje expreso
- *camino*: en la descripción de este procedimiento el término camino hará referencia a un conjunto de viajes espera consecutivos con un único viaje espacial al comienzo de la secuencia ó al final de la misma

Estos conceptos toman importancia debido a que es necesario identificar los caminos que unen dos alelos correspondientes a PET de distintas ciudades. Para poder efectuar el intercambio de toda la información genética en un alelo es necesario que el conjunto de caminos que finalizan en el alelo así como el conjunto de caminos que parten de él, sean idénticos en las soluciones a cruzar.

Funcionamiento del algoritmo:

Sean P_1 y P_2 los individuos progenitores que van a participar en el cruzamiento. En primer lugar se escoge de manera aleatoria un alelo para someterlo al cruzamiento.

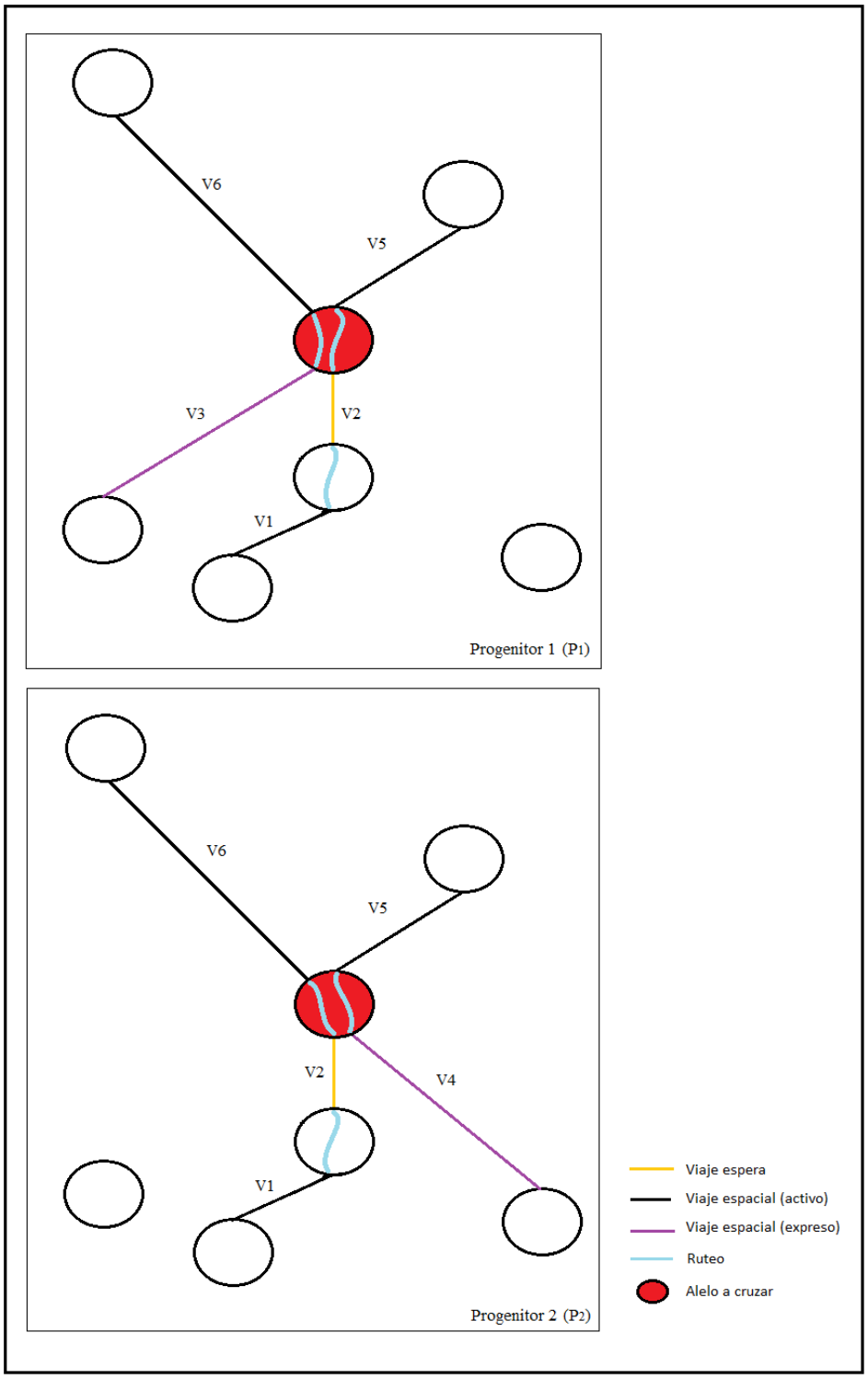

Figura 5.6: Individuos a participar en el cruzamiento

Aplicando la definición de *camino*, descrita anteriormente en esta misma sección, a los progenitores de la Figura 5.6 vemos que los caminos previos en el alelo a cruzar del progenitor P_1 son {V3} y {V2, V1} mientras que los caminos previos de P_2 son {V2, V1} y {V4}.

Para operar sobre las soluciones, de manera de contar con la información nueva por un lado y la antigua por otro, se realiza una copia limpia de ambos individuos

progenitores P_1 y P_2 las cuales llamaremos H_1 y H_2 respectivamente. Para el par (P_1, H_2) se comparan todos los caminos que llegan al alelo sorteado realizando las siguientes acciones:

- aquellos caminos que existen en H_2 pero no en P_1 se eliminan de H_2
- aquellos caminos que existen en P_1 pero no en H_2 se agregan a H_2

Si aplicamos las acciones anteriores a los progenitores P_1 y P_2 y a sus copias H1 y H2 vemos que:

- dado que el camino {V4} no existe en P_1 se procede a eliminarlo de H_2
- el camino {V3} existen en P_1 pero no en H_2 por los tanto se agrega el mismo a esta solución
- como el conjunto de caminos posteriores de ambas soluciones coinciden no hay que aplicar ninguna operación sobre el conjunto de caminos posteriores de H_2

En consecuencia los caminos previos de H_2 resultan ser, luego de aplicadas las modificaciones mencionadas, {V2, V1} y {V3} y los caminos posteriores son {V5} y {V6}. En la Figura 5.7 se muestra de manera grafica los caminos previos y posteriores resultantes de H_2:

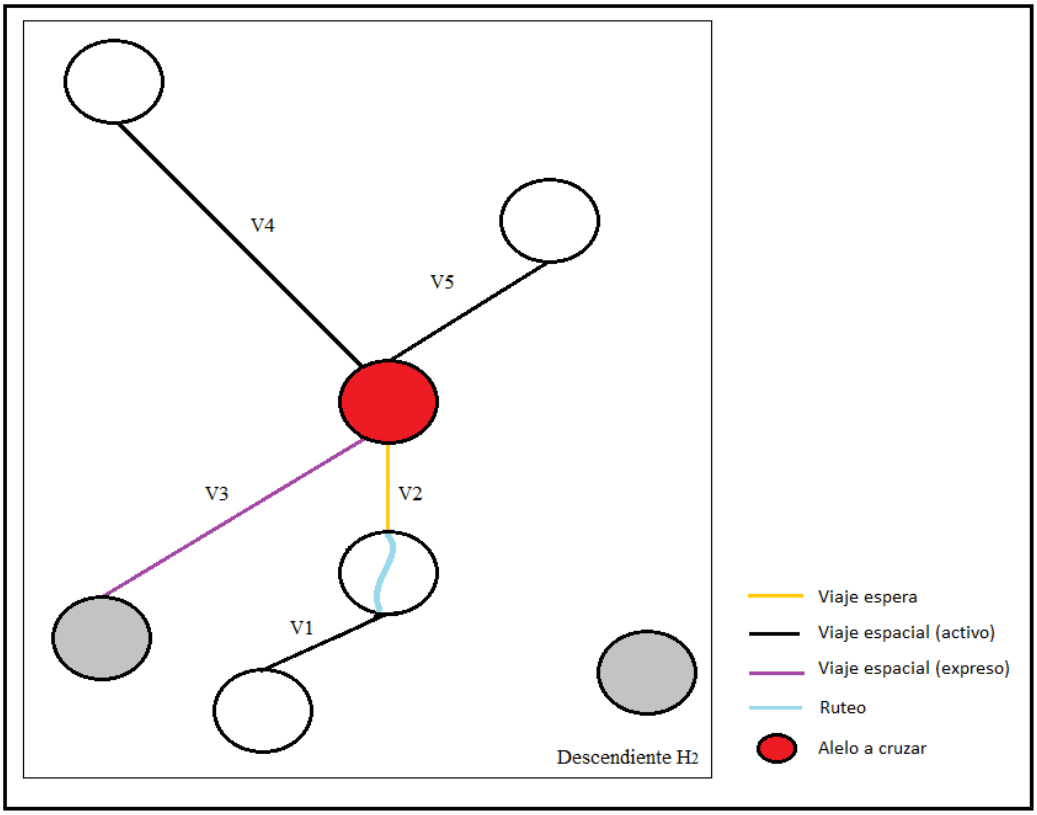

Figura 5.7: Caminos resultantes en H_2

Una vez que se llega a una configuración en la cual el conjunto de caminos previos de H_2 coincide con el conjunto de caminos previos de P_1 y el conjunto de caminos posteriores de H_2 es idéntico al conjunto de caminos posteriores de P_1 se procede a copiar la información genética desde P_1 hacia H_2. En la Figura 5.8 se muestra el resultado de cómo queda el ruteo en el PET que fue sometido al cruzamiento en el individuo H_2:

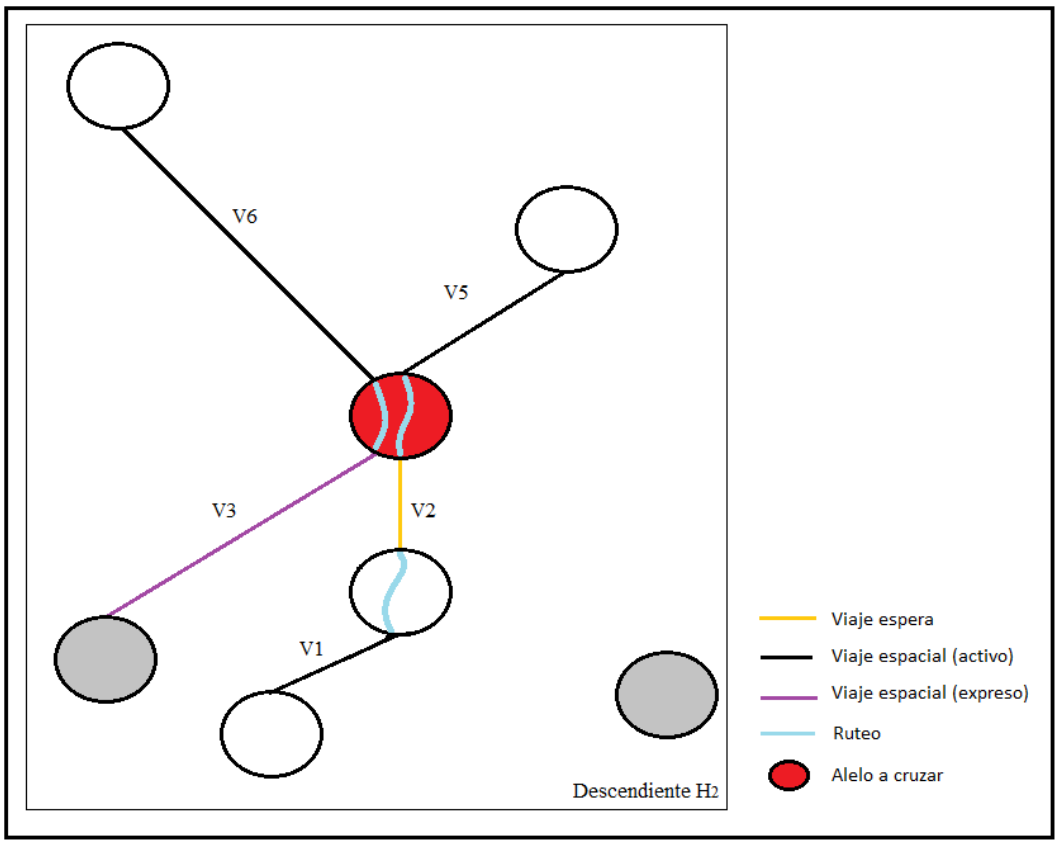

Figura 5.8: Ruteos resultantes en H_2

El procedimiento anteriormente descrito se puede considerar como el paso base de la recursión llevada a cabo por el operador de cruzamiento recursivo. La necesidad de pasos recursivos surge cuando uno o varios alelos se ven afectados por las acciones realizadas sobre el individuo en algún paso anterior. Para poder determinar sobre cuales alelos se deben disparar llamados recursivos, durante el paso base, se va guardando un registro de él o los alelos que resultan afectados por eliminación o creación de nuevos caminos. Existen cuatro escenarios diferentes en los cuales se debe prestar especial atención a la hora de determinar cuál es el alelo afectado del cual se debe guardar registro, en función de si el camino modificado es el previo o el posterior y de si el viaje espacial involucrado es activo o expreso. Si se trata de la modificación (creación o eliminación) de un camino previo y el viaje espacial del camino es un viaje activo o si se modifica un camino posterior y el viaje es expreso, entonces el alelo que se ve afectado es el correspondiente al PET destino de dicho viaje espacial. En cambio, si se trata de la modificación de un camino previo y el viaje espacial del camino es un viaje expreso o si se modifica un camino posterior y el viaje es activo, entonces el alelo que se ve afectado es correspondiente el PET origen de dicho viaje espacial. En la siguiente tabla se resumen los distintos escenarios:

		Viaje Espacial	
		Activo	Expreso
Camino modificado	Previo	PET destino	PET origen
	Posterior	PET origen	PET destino

Figura 5.9: Escenarios

En el caso del ejemplo los alelos afectados se muestran en la Figura 5.8 sombreados en color gris. De manera recursiva se procede a aplicar el mismo algoritmo que el correspondiente al paso base para cada uno de los alelos afectados. Luego, lo explicado para el par (P_1, H_2) se aplica de manera análoga al par (P_2, H_1).

Se hace notar que el tiempo de ejecución de este operador genético dependerá de cuan distintas sean las soluciones a cruzar, dado que se puede llegar a visitar desde un único alelo hasta todos los alelos del individuo. Si se da este último caso (peor caso) se tendrá como resultado que H_1 será exactamente igual a P_2 y H_2 lo será a P_1.

5.4.3 Mutación

El operador de mutación es un operador evolutivo que dota de diversidad a la población de soluciones. La principal razón para incorporar este tipo de operadores radica en tratar de evitar que la búsqueda quede atrapada en torno a óptimos locales, los cuales pueden ser alcanzados a lo largo del proceso evolutivo. Al igual que con los operadores de cruzamiento, los operadores de mutación implementados no siguen el modelo de ningún operador estándar sino que el diseño de los mismos fue creado por el equipo de trabajo de Marco Polo y este se encuentra fuertemente acoplado a lo que es la codificación de la solución escogida. A continuación se detallan los distintos operadores de mutación implementados

5.4.3.1 Mutación Expresos Intercambiables

Esta mutación busca modificar viajes expresos que son llevados a cabo por vehículos con sus respectivos turnos en un determinado individuo.

A continuación se detalla el funcionamiento del operador de mutación. Sean los viajes V_{wx} y V_{yz} dos viajes expresos de un individuo, tal que el primero hace referencia al viaje que parte del $alelo_w$ y arriba al $alelo_x$, mientras que el segundo representa el viaje que parte del $alelo_y$ y arriba al $alelo_z$. Si ambos viajes son intercambiables, es decir que, se cumple simultáneamente que:

- El tiempo necesario para ir desde la ciudad asociada al $alelo_w$ hacia la ciudad asociada al $alelo_z$ es menor a la diferencia horaria entre ambos alelos

- El tiempo necesario para ir desde la ciudad asociada al $alelo_y$ hacia la ciudad asociada al $alelo_x$ es menor a la diferencia horaria entre ambos alelos

Entonces se realizan las siguientes acciones sobre el individuo:

- Se agrega el viaje expreso V_{wz}
- Se agrega el viaje expreso V_{yx}
- Se elimina el viaje expreso V_{wx}
- Se elimina el viaje expreso V_{yz}

Para ilustrar el funcionamiento del operador de mutación definido, se muestra a continuación un ejemplo de aplicación del mismo. En dicho ejemplo se definen cuatro ciudades: Montevideo, Solymar, El Pinar y Salinas. El individuo al cual se le aplicará el operador de mutación presenta un viaje expreso que parte desde Solymar a las 7 de la mañana y arriba a El pinar a las 7:50 am y otro que tiene origen en Salinas a las 7:10 am y destino Montevideo a las 8:00 am. Supóngase que realizar el trayecto Solymar - Montevideo requiere de una duración de 45 minutos y para trasladarse desde Salinas a El pinar son necesarios al menos 15 minutos de viaje. Esto significa que los viajes expresos son intercambiables por lo tanto el operador de cruzamiento puede ser aplicado sobre ellos. En la Figura 5.10, puede observarse la zona del individuo que será sometida a mutación y el resultado de la aplicación del operador genético.

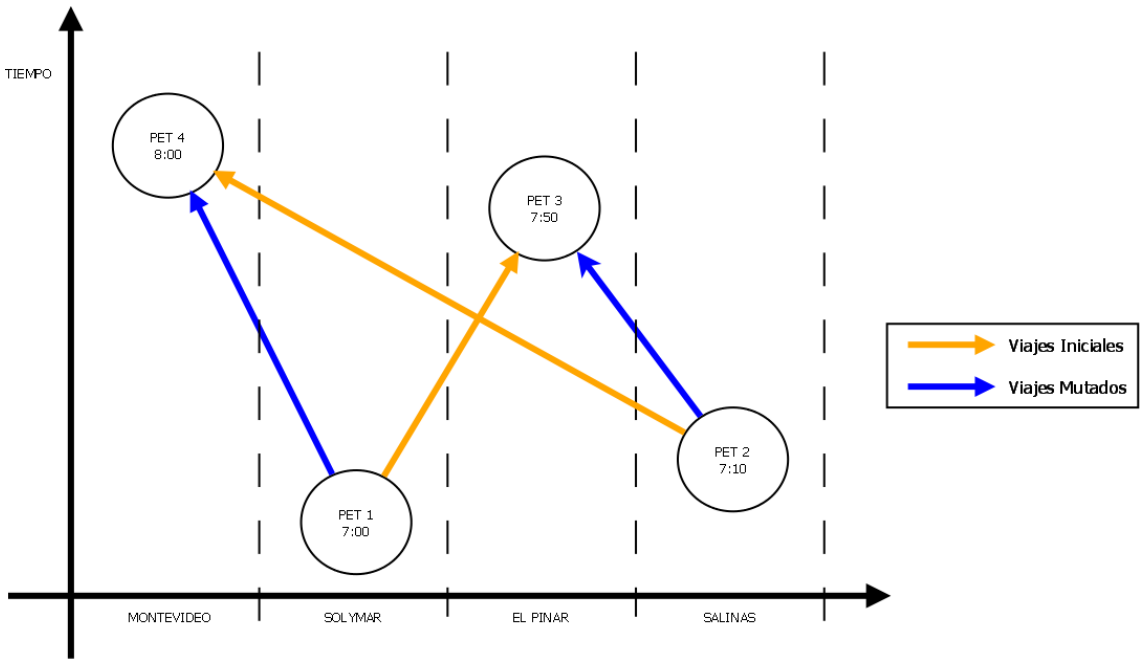

Figura 5.10: Mutación expresos intercambiables

5.4.3.2 Mutación Separar Turnos

Para comprender esta mutación en primer lugar se debe entender el procedimiento definido para cortar un recorrido. Cortar un recorrido da como resultados dos recorridos nuevos. Existen dos formas de realizar el corte:

- Caso 1: El corte propiamente dicho se realiza en un record. Lo que va a suceder es que el record en donde se hizo el corte desaparecerá y se crearán dos nuevos, uno que pertenece al destino del último viaje del primer nuevo recorrido generado y otro que pertenece al inicio del primer viaje del segundo nuevo recorrido generado.

- Caso 2: A un recorrido se le eliminan una cierta cantidad de viajes consecutivos. Aquí se deberán eliminar los records intermedios que no forman parte de ninguno de los dos nuevos recorridos.

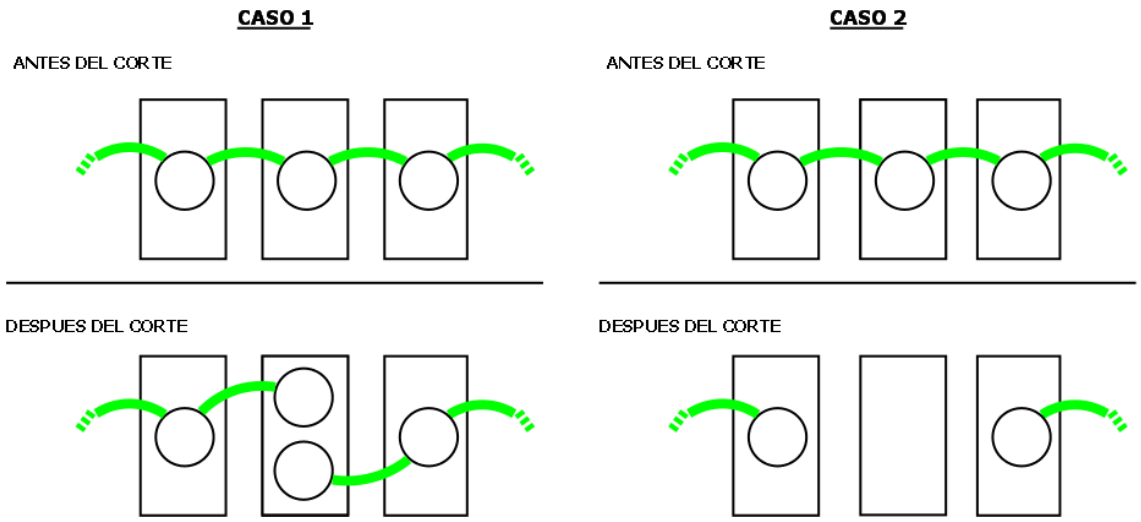

Figura 5.11: Corte de recorridos

Entendido el procedimiento de corte de un recorrido se procede a explicar la mutación en sí. Trabajando sobre el individuo hay que tener presente que el recorrido del turno está vinculado con el del vehículo, por lo tanto, si se realiza un corte en alguno de estos dos tipos de recorrido el otro se va a ver afectado. Esta mutación maneja dos tipos de cortes: el corte entre turnos y el corte en turnos. Para explicar ambas técnicas usaremos el ejemplo de recorrido de la Figura 3.7 presentada en la sección 3.2.3:

1. Corte entre turnos:

Este corte se realiza entre dos turnos consecutivos que estén asociados al mismo vehículo. Los dos recorridos de turnos quedan intactos, pero el recorrido del vehículo queda dividido en dos nuevos recorridos.

Al querer hacer el “corte entre turnos” se puede dar el caso en que el alelo destino del último viaje para el primer turno sea el mismo que el alelo origen del primer viaje para el segundo turno como sucede en el tercer alelo del ejemplo entre los turnos A y B. Aquí se deberá aplicar la técnica de corte del caso 1 mencionada anteriormente. A continuación se puede ver el resultado de ejecutar el “corte entre turnos” entre los turnos A y B

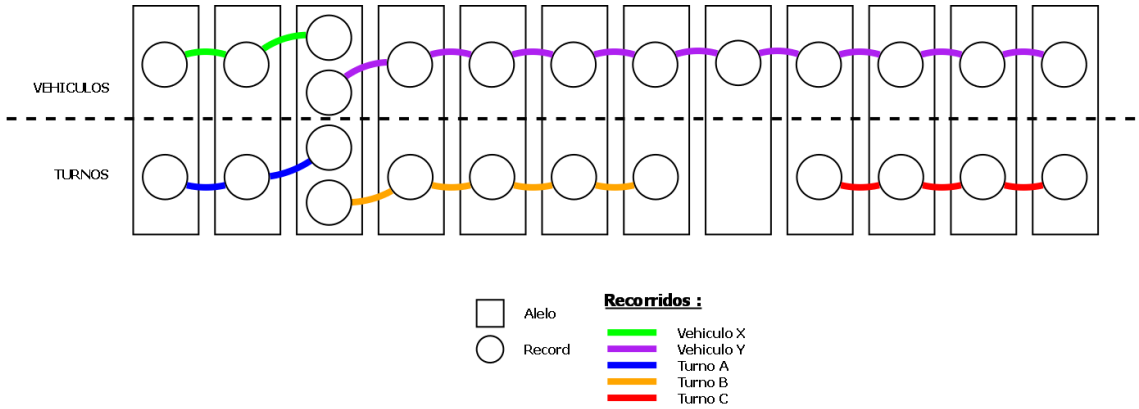

Figura 5.12: Corte entre turnos aplicado a los turnos A y B

El otro caso que puede darse al querer hacer este corte es que los turnos estén separados por uno o más viajes esperas como sucede en el octavo alelo del ejemplo. Aquí se deberá aplicar la técnica de corte del caso 2 mencionada anteriormente de manera de eliminar los viajes espera que el vehículo hacía entre los turnos y tendremos como resultado que:

- El primer nuevo recorrido de vehículo empezará en el mismo lugar en que comenzaba el recorrido de vehículo original antes de realizar el corte
- El primer nuevo recorrido de vehículo terminará en donde termina el primer turno
- El segundo nuevo recorrido de vehículo empezará en donde empieza el segundo turno
- El segundo nuevo recorrido de vehículo terminará en el mismo lugar en que terminaba el recorrido de vehículo original antes de realizar el corte

A continuación se puede ver el resultado de ejecutar el “corte entre turnos” entre los turnos B y C

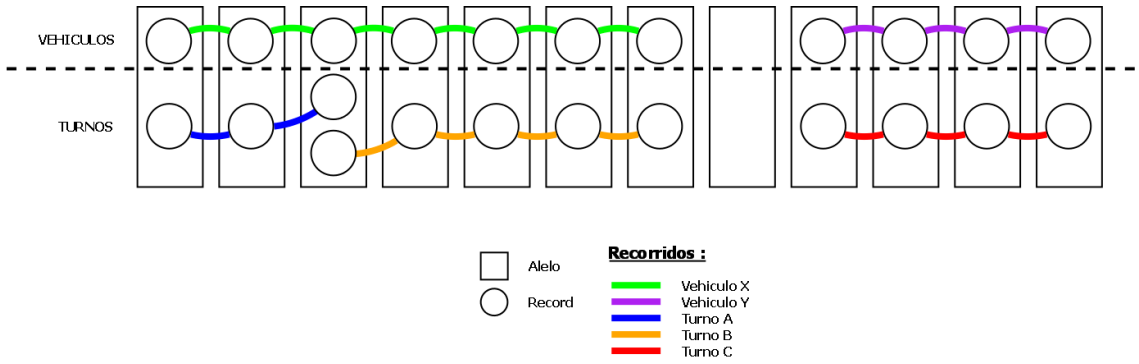

Figura 5.13: Corte entre turnos aplicado a los turnos B y C

2. Corte en turnos:

Este corte se realiza en medio de un turno. Tanto el recorrido del turno como el del vehículo quedan divididos en dos nuevos recorridos.

En esta oportunidad se debe seleccionar el turno a cortar y el record en donde se efectuará el corte. De esta manera se obtendrán dos nuevos recorridos de turnos. Si el turno original tiene N viajes, entonces su recorrido poseerá N+1 records. Para cortar este turno se podrá elegir entre N-1 posibles cortes. Estas posibilidades son en el segundo record, el tercero y así sucesivamente hasta el penúltimo. Es importante notar que si se quisiera hacer el corte en el primer record o en el último estaríamos en el caso del "corte entre turnos".

Una vez cortado el turno separaremos el recorrido del vehículo en dos. Está claro que el corte del vehículo se realizará en el mismo alelo que se hizo el corte del turno. Luego de realizado este corte, tanto la cantidad de vehículos como la cantidad de turnos en el individuo se incrementarán en 1. Siguiendo con el ejemplo, a continuación se puede ver el resultado de ejecutar un corte en el turno B:

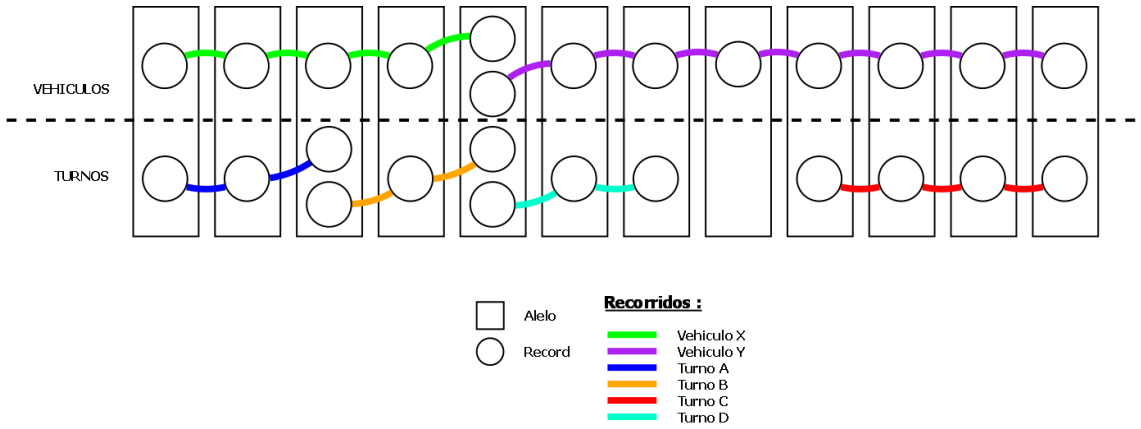

Figura 5.14: Corte en turnos aplicado al turno B

Un detalle interesante es que, luego de realizar este tipo de cortes, los turnos pueden terminar o empezar (según sea el caso) con viajes expresos o esperas. Dependiendo de la intención con que se realizó el corte podríamos llegar a querer quitar estos viajes.

5.4.3.3 Mutación Unir Turnos

Análogamente que en el caso de los cortes, para comprender esta mutación, en primer lugar se debe entender el procedimiento definido para unir un recorrido. Para unir dos recorridos se debe contar con dos recorridos iniciales y el resultado de la unión será un único recorrido.

En adelante llamaremos al alelo asociado al primer record de un recorrido el "alelo inicial" y al alelo asociado al último record de un recorrido el "alelo final". Estos alelos, como todos, tienen asociados un horario y una ciudad, los cuales se corresponden con el PET al cual están representando. Una restricción para que la unión entre dos

recorridos pueda llevarse a cabo es que el horario del alelo final del primer recorrido debe ser menor al horario del alelo inicial del segundo recorrido. Cumpliendo con esta restricción pueden generarse dos casos:

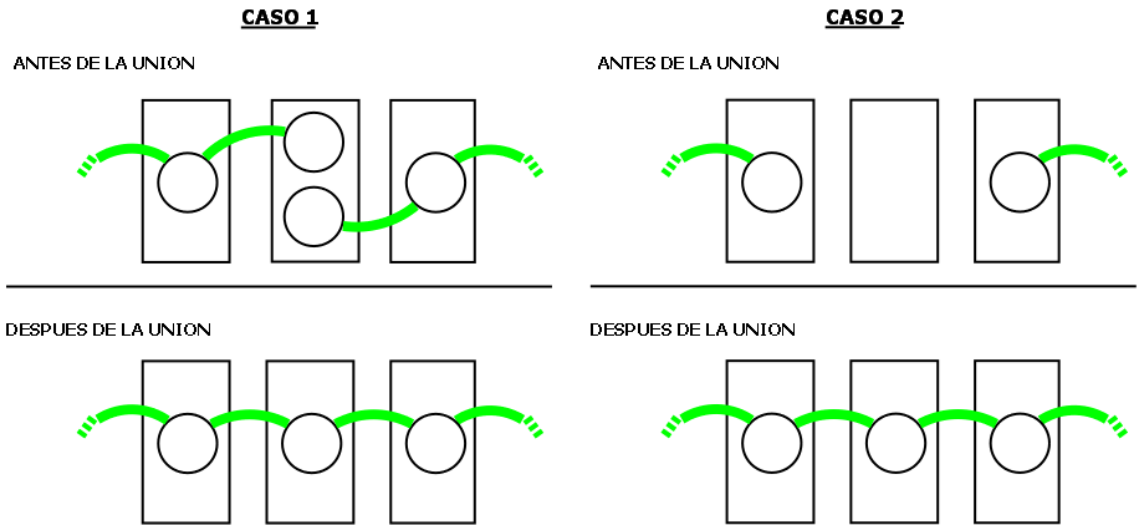

Figura 5.15: Unión de recorridos

Como se ve en la Figura 5.15, en el caso 1, el alelo inicial del primer recorrido es el mismo que al alelo final del segundo recorrido y, en el caso 2, los alelos iniciales y finales de cada recorrido no son los mismos. En el caso 2 la unión se podrá realizar en función de si las ciudades asociadas a los alelos inicial y final del primer y segundo recorrido son la misma o no. En caso de que ambos alelos estén asociados a la misma ciudad la unión podrá efectuarse sin problemas, teniendo que agregar todos los viajes espera que correspondan según la diferencia horaria de cada alelo. Por otro lado si las ciudades son diferentes, para que se pueda realizar la unión, debe cumplirse la condición de que exista un viaje expreso entre dichos alelos. En caso contrario no podrá llevarse a cabo la unión.

Entendido el procedimiento de unión entre dos recorridos se procede a explicar la mutación en sí. Para realizar la unión entre dos vehículos es necesario contar con dos recorridos de vehículos. Es necesario tener presente que cada uno de estos recorridos tendrá asociado uno o más recorridos de turnos. Luego de corroborar si dos recorridos cumplen las condiciones necesarias para que se pueda llevar a cabo su unión, resta decidir si queremos unir el último turno del primer vehículo con el primer turno del segundo vehículo o no. En la Figura 5.16 se muestran ambas posibilidades:

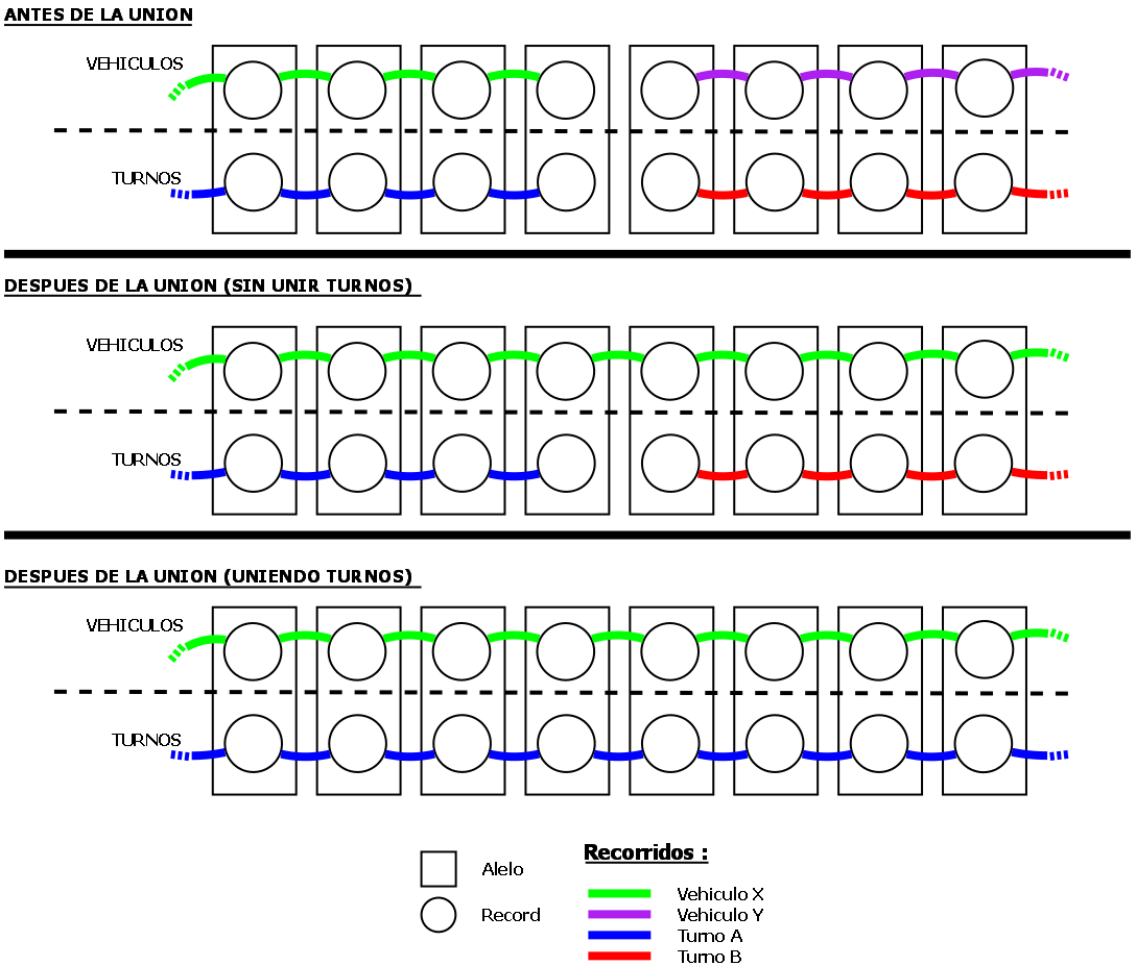

Figura 5.16: Mutación unir turnos

Es claro que unir dos recorridos de vehículos reduce en uno la cantidad de vehículos total en el individuo y dependiendo de si se unen los turnos o no reducen también la cantidad de turnos en una unidad o no.

5.5 Población inicial

La generación de la población inicial es uno de los puntos clave que, en definitiva, terminará determinando el comportamiento final del algoritmo. Es importante generar suficiente diversidad en los datos de entrada para no perder la oportunidad de utilizar configuraciones que puedan llevar al algoritmo a un resultado cercano al óptimo. Por otro lado, como puede verse hasta aquí, tanto en el problema a resolver como en la codificación diseñada, estamos frente a un problema complejo para el cual no será sencillo generar soluciones iniciales factibles.

Es por esto que se presenta la necesidad de aislar el problema de generación de soluciones iniciales en un módulo independiente dedicado exclusivamente a esta tarea (Anexo D). La idea detrás de este módulo es contar con métodos flexibles que permitan aplicar distintas técnicas de construcción de soluciones y hasta combinarlas entre sí para lograr la mayor diversidad y calidad posible en la generación inicial del algoritmo.

En este punto del documento es necesario agregar información a la definición de microturno (Sección 3.1.2.2). Los microturnos son turnos especiales que se generan para poder construir más fácilmente individuos factibles en la población inicial. Estos turnos tienen la ventaja de no tener que cumplir reglas laborales muy estrictas, pero por otro lado tienen la desventaja de ser muy costosos, dado que sufren la penalización habitual que sufre cualquier turno que dure menos que la duración definida como "normal" para una jornada laboral. Esta penalización se expresa en el sentido de que se deberá abonar el jornal completo por más de que el turno tenga una duración reducida. Consecuentemente los individuos generados con microturnos serán factibles, en cuanto a que respetan las reglas laborales, pero a su vez serán también muy costosos y, dada su escasa duración, cubrirán poca cantidad de viajes en relación al costo que implica realizarlos, con lo cual tenderán a desaparecer a lo largo de las ejecuciones.

En el proyecto Marco Polo se implementan varios modos de generación de soluciones iniciales diferentes, basados en ideas constructivas para generar las soluciones. Como ya se anticipó en el Capítulo 4, al momento de elaborar la población inicial, se utilizan internamente algunas de las técnicas heredadas del proyecto predecesor (2) en el momento del armado de los recorridos. A continuación se enumeran los distintos modos implementados:

- Modo 1: en esta técnica se generan soluciones cuyos servicios contienen un solo turno cada uno. En este modo no se chequea el cumplimiento de las reglas laborales. Su implementación se detalla en la sección D.5.3.1.

- Modo 2: esta técnica es similar a la anterior solo que cuenta con un valor entero configurable (N) el cual indicará la cantidad de turnos que deberá contener cada servicio perteneciente a las soluciones generadas. En este caso tampoco se chequea el cumplimiento de las reglas laborales. Su implementación se detalla en la sección D.5.3.2.

- Modo 3: esta técnica genera soluciones cuyos servicios contienen microturnos compuestos únicamente por un viaje. En esta técnica se puede elegir si se chequea o no el cumplimiento de las reglas laborales mediante un parámetro en el archivo general de configuración Conf.cfg (6.1.1). Su implementación se detalla en la sección D.5.3.3.

- Modo 4: esta técnica también elabora soluciones con microturnos pero, en este caso, serán de una duración mayor de manera que se aproximen más a la duración de un turno convencional. Al igual que en la técnica anterior, se puede elegir si se chequea o no el cumplimiento de las reglas laborales mediante un parámetro en el archivo general de configuración Conf.cfg (6.1.1). Su implementación se detalla en la sección D.5.3.4.

La implementación de los distintos modos se hizo de manera de que se generen siempre soluciones iniciales que aseguren la propiedad de cubrimiento total de los viajes activos. En el Anexo D se detalla la implementación de cada uno de los modos.

Capítulo 6: Marco Polo

En este capítulo se presentarán los distintos módulos definidos en el proyecto Marco Polo y como estos interactúan con el algoritmo genético brindándole los datos de la realidad modelados de manera que sean fácilmente accesibles cuando se requieran.

Cuando se quiere modelar la realidad hay que decidir qué información se necesita extraer de ella para luego diseñar tanto las estructuras que contendrán la información a almacenar como las funciones que se ofrecerán a partir de dicha información. Dado que el problema a resolver maneja y genera un gran volumen de datos, uno de los principales objetivos que nos planteamos, al momento de diseñar las estructuras, fue el de lograr eficiencia en el almacenamiento de estos datos. A su vez la performance de las operaciones que ofrecen los módulos está fuertemente ligada a como la información es organizada. Los tiempos de acceso a los datos tienen que permanecer en órdenes bajos y para eso es necesario apelar a técnicas que reduzcan estos tiempos. A veces la redundancia puede servir pero hay que ser cuidadosos con su manutención. Los índices sirven para implementar las búsquedas y tener diferentes índices sobre un conjunto de datos da la posibilidad de elegir en qué orden recorrer o buscar elementos de ese conjunto.

Los principales módulos de Marco Polo surgieron para proveer al algoritmo genético de los datos necesarios de la realidad para poder resolver el problema así como de distintas funcionalidades que implementan servicios útiles para el algoritmo. Toda la información sobre los viajes, los activos con sus respectivos datos de origen y destino, así como los posibles expresos, para los cuales son necesarios los datos de distancias y tiempos entre los distintos lugares geográficos, son administrados por un módulo denominado gran grafo (en adelante GG). Por otro lado se encuentra la información relativa a las reglas laborales, que regulan la actividad y los descansos de los trabajadores, la cual es cargada y administrada en el módulo de reglas laborales (en adelante MRL). La implementación del algoritmo genético está inmersa también en un módulo, llamado AE, el cual encapsulará la comunicación con Mallba (16), el motor de resolución que implementa la metaheurística. Por otro lado, se decide aislar la generación de soluciones iniciales por ser esta una tarea de elevada complejidad. El módulo encargado de la generación de soluciones iniciales (en adelante GSI) también se alimenta de los datos provistos por el GG y el MRL para entregarle al AE la población que se usará en la primer generación del algoritmo. En el GSI se implementan distintos modos de generación de soluciones iniciales. Para poder independizar el armado de estos distintos modos de la implementación de cada técnica en sí, se cuenta con el módulo de Workflow (en adelante WF) el cual, a partir de un esquema de ejecución dado, invoca las distintas funciones implementadas. La Figura 6.1 muestra un esquema con los distintos componentes de Marco Polo y como estos se relacionan.

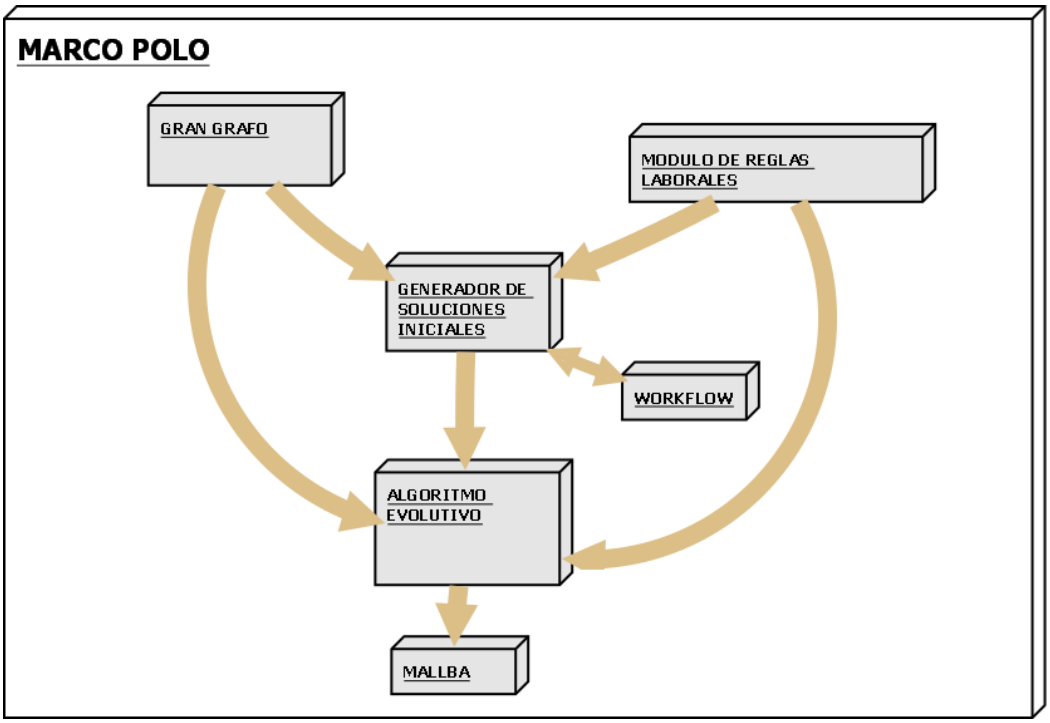

Figura 6.1: Diagrama de componentes de Marco Polo

Al comienzo de la ejecución, en el programa principal, se invoca al constructor del GG para crear la única instancia del grafo que existirá a lo largo de toda la ejecución. Esta instancia mantiene en memoria los distintos datos de la realidad, los cuales son levantados a partir del archivo que contiene la especificación del problema. A continuación se instancia el MRL el cual también mantendrá en memoria la información referente a las reglas laborales. Una vez creadas estas estructuras comienza la ejecución del algoritmo genético. En la primera iteración se invocará al GSI, el cual participará en la ejecución esta única vez construyendo la población de la primera generación del algoritmo genético, con lo cual tendrá un ciclo de vida mucho más corto que el del resto de los módulos. Para cumplir su cometido el GSI invocará al constructor del módulo WF creando una instancia particular para organizar y llevar a cabo su ejecución. Una vez que se cuenta con la generación inicial comienzan las iteraciones habituales de un algoritmo genético, en donde se aplican los distintos operadores, hasta que se cumple un determinado criterio de parada, que por lo general suele ser un número prefijado de generaciones.

6.1 Arquitectura y diseño

Marco Polo fue diseñado de manera modular e implementado en C++ orientado a objetos. La idea detrás de la modularización de los distintos servicios y funcionalidades ofrecidas se basa en generar un software fácilmente extensible y del cual a su vez se puedan extraer piezas concretas (módulos) para modificarlas, re-utilizarlas en otro contexto y hasta intercambiar un módulo por otro siempre y cuando se respeten las operaciones que este ofrece.

6.1.1 Configuración general en Marco Polo

Debido a la gran cantidad de datos configurables que se incluyen en un algoritmo genético y dado la inmensa variedad de parámetros que se puede definir para modelar un problema como el que se intenta resolver en el presente proyecto, surge la necesidad de organizar y centralizar el acceso a toda esta información en un único punto. En Marco Polo se concentra la configuración de todo el algoritmo en una clase singleton *ConfiguracionGlobal*. Esta clase parsea, al inicio de la ejecución y previo a la creación del GG, el archivo Config.cfg el cual contiene las configuraciones y parametrizaciones del sistema de cualquier índole. De esta manera, cada uno de los módulos de Marco Polo que necesite acceder a datos definidos por el usuario, deberá solicitarle estos datos a la clase *ConfiguracionGlobal*. El contenido del archivo Config.cfg se detalla en la sección A.4.

Esta decisión de diseño aporta le aporta legibilidad al código y una fácil localización y visualización de la configuración con la cual se está ejecutando el algoritmo en un determinado momento. Se pueden almacenar diversas configuraciones personalizadas simplemente definiendo distintos archivos de parámetros e invocando uno diferente cada vez. Adicionalmente se torna sumamente sencillo definir un nuevo parámetro.

6.1.2 AE: módulo del algoritmo genético

La implementación de todo lo necesario para que se ejecute el algoritmo genético se encuentra en el módulo AE. La siguiente Figura 6.2 muestra su diagrama de clases:

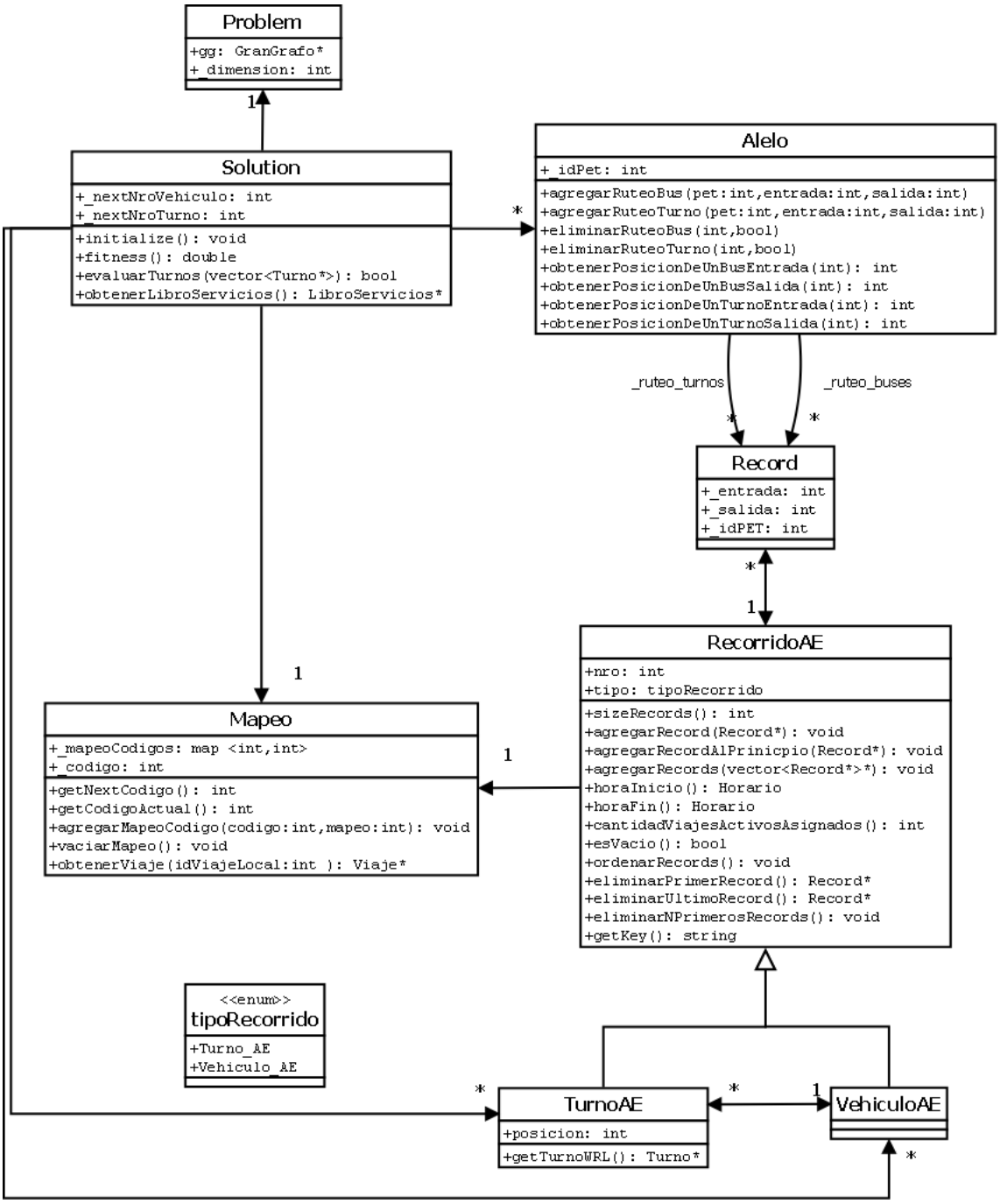

Figura 6.2: Diagrama de clases del módulo AE

Este módulo representa al problema instanciado y es asistido por un motor de resolución que implementa distintas metaheurísticas. Este motor provee al programador de esqueletos algorítmicos que implementan métodos genéricos y, para utilizarlo, uno debe crear una instancia concreta e implementar determinadas interfaces y métodos requeridos. A su vez, el algoritmo genético, durante el transcurso de toda su ejecución, necesitará consumir operaciones que él mismo no es capaz de resolver, como ser, las operaciones asociadas a la realidad del problema. La visibilidad se define entonces desde el AE hacia los otros módulos de Marco Polo y no de manera inversa. Con esto se ve claramente que el algoritmo es un cliente consumidor de las operaciones que el resto de los módulos ofrecen.

Mallba (16) es una biblioteca, reconocida en el ámbito académico, que implementa esqueletos algorítmicos para resolver problemas de optimización combinatoria. Un esqueleto es un procedimiento genérico que le permite al usuario crear instancias concretas de su problema en base a instanciar y completar la implementación de los distintos métodos provistos. Su implementación está compuesta por un conjunto de clases en C++ llamadas “clases provistas”, las cuales implementan aspectos internos que no tienen que ver con el problema concreto por lo que no deberían ser modificadas por el usuario, y otro conjunto denominadas “clases requeridas”, las cuales representan aspectos concretos del problema a resolver y deben ser instanciadas e implementadas por el usuario para poder hacer uso de la biblioteca. Estos conjuntos de clases representan a las entidades que participan en el método de resolución que está implementando un esqueleto. Mallba ofrece métodos exactos, métodos heurísticos y métodos híbridos para la resolución de distintos problemas de optimización y ofrece además implementaciones secuenciales y distribuidas. La principal motivación para usar una biblioteca de estas características es que le ofrece al usuario final un nivel de abstracción que permite que este solo deba preocuparse por elegir un método de resolución, y su correspondiente esqueleto algorítmico, sin tener que preocuparse por aspectos relacionados a la implementación del algoritmo en sí con las dificultades que esto conlleva. Adicionalmente el grupo del proyecto Marco Polo tiene experiencia previa en el uso de esta herramienta y fue también la biblioteca elegida por el proyecto predecesor (2) para el desarrollo de su propio sistema. Por otro lado Mallba cuenta con la característica indispensable de ser de código libre y abierto.

En Marco Polo se utiliza el esqueleto de la metaheurística Algoritmos Genéticos (*newGA*) en su versión secuencial. Las clases requeridas a implementar para instanciar dicho algoritmo son las clases *Problem*, *Solution*, *Crossover* y *Mutation*.

6.1.3 GG: gran grafo

El gran grafo representa la malla de viajes existentes en la realidad a partir de la cual luego surgirán las distintas soluciones posibles. La información más importante almacenada dentro de este módulo es el conjunto de viajes activos, con sus lugares geográficos y horarios de partida y arribo, además de todo lo relativo a distancias, en términos de kilómetros y de tiempo, entre los distintos lugares.

En este módulo se representa la realidad como un grafo en donde cada arista es un viaje y cada nodo un punto del espacio-tiempo (PET). Estas dos entidades son representadas en nuestro proyecto por medio de clases, en donde la clase PET tendrá entre sus atributos la lista de punteros a los viajes que salen y los que entran a dicho PET. A su vez, la clase viaje tendrá un puntero hacia el PET origen y otro al PET destino. Con esto logramos armar caminos doblemente encadenados.

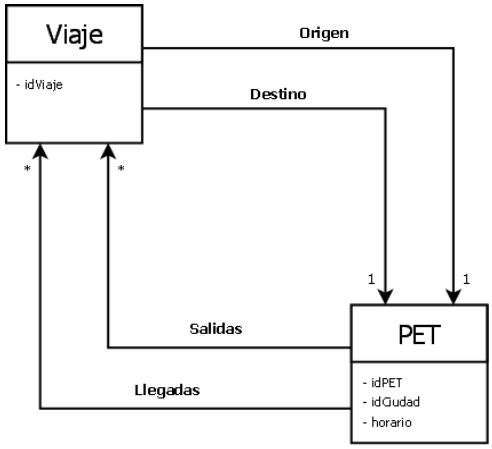

Figura 6.3: Relación entre Viaje y PET

Se toma la decisión de que en el GG solo se instanciarán aquellos PET que tengan al menos un viaje activo que parte o llega al mismo. A partir de estos PET se crearán todos los viajes expresos y espera posibles para utilizar luego en el armado de las soluciones. Toda esta información es mantenida en memoria durante todo el ciclo de vida del GG

Las estructuras de datos utilizadas en el GG fueron diseñadas de manera que el acceso a la información se resuelva, en todos los casos, en operaciones de orden uno. Este diseño surge motivado por la elevada cantidad de iteraciones que componen la ejecución de un algoritmo genético, en cada una de las cuales, se requerirá en repetidas ocasiones este tipo de información.

En el Anexo B se documenta detalladamente el funcionamiento de este módulo.

6.1.4 MRL: módulo de reglas laborales

Existen dos grandes recursos en la realidad en la cual está inmersa el problema planteado en el presente proyecto: los vehículos y los recursos humanos. Debido a la complejidad que posee por sí sola la administración de los recursos humanos, resulta necesaria la creación de un módulo especializado en esta realidad que se abstraiga del problema de planificación planteado. El MRL es quien implementa esta abstracción, encapsulando el problema de cómo la empresa de transporte puede hacer uso de los recursos humanos respetando lo dispuesto por la normativa vigente y los acuerdos gremiales alcanzados. Esto dota de una propiedad por más interesante al MRL y es que el mismo puede ser utilizado independientemente de la estrategia de planificación que se elija. El módulo presenta una interface que lo hace totalmente integrable a cualquier proyecto relacionado con la temática aquí tratada.

En el caso concreto de Marco Polo, en el MRL, se modelan las reglas que reflejan la realidad del caso de estudio con el que se cuenta para realizar las pruebas del algoritmo. Concretamente se utilizan las mismas reglas que se modelan en el módulo de reglas laborales del PG42 (2), las cuales son:

- Duración mínima del jornal: 8 horas

- Los descansos deben satisfacer una de las siguientes condiciones:
    - Tres descansos de 15 minutos
    - Dos descansos de 20 minutos
    - Un descanso de 50 minutos

Para modelar con más precisión la realidad, se definen parámetros que permiten flexibilizar la duración de estos descansos. Esto se debe a que los viajes, al momento de ser ejecutados, pueden no cumplir con los tiempos teóricos establecidos, a causa de factores externos al problema (embotellamientos, accidentes, etc.).

Dado lo complejas y restrictivas que son estas reglas es que surge el concepto de microturno (Sección 3.1.2.2) el cual lleva a tener que definir otro conjunto de reglas, también válidas, con características menos exigentes para su cumplimiento. Las reglas laborales definidas para los microturnos son:

- La duración de un microturno debe ser menor o igual a 240 minutos (4 horas)
- Pueden no existir descansos en un microturno
- La suma de los minutos de todos los descansos existentes en un microturno no puede superar la mitad de la duración total del mismo

La manera en que se modelan estas reglas internamente en el MRL hace que se puedan definir distintos conjuntos de reglas y que un turno sea factible si cumple alguno de todos estos conjuntos. En el Anexo C se brindan los detalles de implementación y la sintaxis del modelado de las distintas reglas y conjuntos de reglas en el MRL.

6.1.5 GSI y WF: módulos de soluciones iniciales y de workflow

En el GSI las soluciones se construyen de manera incremental, tomando uno tras otro los viajes activos disponibles y armando, con diversos criterios, los recorridos que compondrán los servicios. La primer idea que surge, al comenzar a construir un módulo independiente para la generación de soluciones iniciales, es la de poder incorporarle distintas técnicas de construcción para alcanzar dichas soluciones. A su vez se desea incorporar aleatoriedad a la construcción de los distintos recorridos para poder lograr una población diversa. Principalmente para orquestrar la secuencia de acciones a ejecutar en el GSI, es que surge la idea del módulo de Workflow (WF). Además era necesario contar con un módulo maleable y versátil que brindara la posibilidad de extenderlo fácilmente con nuevas instrucciones que pudieran surgir.

A partir del patrón general de cómo se quieren construir los libros de servicios de la población inicial, junto con instrucciones que implementan las distintas técnicas a aplicar para ir formando soluciones parciales, se genera una instancia del Workflow en el cual se define el orden en que se desean ejecutar dichas instrucciones y la cantidad de veces que se quieren repetir ciertas iteraciones. Lo interesante del módulo WF es que permite que el programador centre el foco de atención en la implementación de las instrucciones y condiciones que considere necesarias y luego, en una etapa posterior de pruebas, maneje la configuración y el armado del conjunto de instrucciones y condiciones según lo requiera el caso particular. A su vez se cuenta con

una herramienta que brinda la posibilidad de introducir cambios y probar diferentes escenarios.

Se puede encontrar documentación detallada del GSI en el Anexo D y del WF en el Anexo E.

Capítulo 7: Experimentos y resultados

7.1 Introducción

En este capítulo se presenta la calibración de los parámetros necesarios para correr el algoritmo y las ejecuciones del mismo sobre los escenarios dados.

La calibración fue realizada sobre un caso de prueba reducido el cual se considera que es representativo de la realidad del problema planteado. Posteriormente se presentan las ejecuciones, usando los valores antes obtenidos, sobre el escenario definido por el Departamento de Investigación Operativa y el escenario dado por la empresa de Transporte Público.

Con el plan de pruebas se buscar evaluar el desempeño del algoritmo desarrollado tanto en un caso reducido de la realidad como en un caso real. Evaluar el desempeño incluye no solamente observar la calidad de las soluciones que el mismo genera, sino que también se busca apreciar el costo computacional que este requiere para procesar instancias del problema de diferentes dimensiones.

El presente capítulo está subdividido en tres secciones bien diferenciadas pero cada una de ellas es necesaria para poder abordar la sección que le prosigue. En la primera sección (Sección 7.2) se define el plan de pruebas mientras que en la segunda (Sección 7.3) se presenta el estudio computacional del algoritmo. Finalmente en la última sección del capítulo (Sección 7.4) se realiza un estudio comparativo entre Marco Polo y los resultados previamente conocidos para el problema atacado.

Todas las pruebas presentadas en este capítulo fueron realizadas en un equipo modelo *HP 6730b* con procesador *Intel Core 2 Duo*, *CPU P8400* de 2.26 GHz y 2,27 GHz y 2GB de memoria RAM sobre un sistema operativo *Ubuntu 11* de 32 bits.

7.2 Plan de pruebas

7.2.1 Casos de estudio

Los casos de estudio a utilizar nos fueron provistos por el Departamento de Investigación Operativa y se corresponden con un caso reducido elaborado en la facultad de Ingeniería, el cual ha sido denominado como *Caso Héctor*, y con un caso real de la empresa de transporte de pasajeros COPSA. El caso de estudio Héctor consta de 66 viajes en una flota distribuida en 3 ciudades mientras que el caso de estudio COPSA cuenta con 1069 viajes realizados recorriendo más de 180 ciudades.

7.2.2 Parámetros a calibrar

Los parámetros a calibrar son el tamaño de la población y las distintas probabilidades de cruzamiento y mutación para los operadores implementados. Es importante mencionar que en el caso del operador de cruzamiento será ejecutado únicamente el operador de Cruzamiento Recursivo de Subgrafos dado que, como ya se mencionó en

la sección 5.4.2, este incluye como caso particular a la situación implementada en el Cruzamiento en un Alelo.

Dada la cantidad combinaciones posibles que se producen, debido al gran número de parámetros a calibrar, se ejecutaron informalmente pruebas previas con el fin de obtener una idea general del impacto de cada operador para poder acotar el rango de valores posibles en cada caso. A continuación se presenta la selección de valores de tamaño y probabilidades para cada parámetro:

	Valor 1	Valor 2	Valor 3
Población	30	50	-
Cruzamiento	0,2	0,5	0,9
Mutación expresos intercambiables	0,1	0,3	-
Mutación unir turnos	0,2	0,5	0,9
Mutación separar turnos	0,1	0,3	-

Tabla 7.1: Selección de valores

El proceso de calibración consta de 72 ejecuciones las cuales se corresponden a todas las combinaciones posibles que se pueden obtener con los valores presentados (producto cartesiano). En la sección de resultados (Sección 7.3) se presenta, para cada una de estas combinaciones, el mejor costo obtenido luego de tres ejecuciones independientes y el tiempo total empleado en dichas ejecuciones. Para cada ejecución se fijó el número de generaciones en 1000, la semilla en 123456, se usaron reglas laborales (las especificadas en la Sección 6.1.4) y el modo 4 de generación de soluciones iniciales por ser la que aporta más diversidad al comienzo de la evolución. Se hará un análisis comparativo entre los mejores resultados obtenidos en cuanto al costo de la solución, de acuerdo a la función definida en la sección 5.2, y su costo computacional, en términos del tiempo de ejecución, con el objetivo de lograr un compromiso entre ambos valores.

Se realizará además un estudio del costo computacional de cada operador en base al tiempo empleado por cada uno de ellos. Para esto se ejecutarán las siguientes pruebas:

Prueba de operadores	cruza-miento	mutación exp.inter.	mutación unir turnos	mutación sep.turnos
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Tabla 7.2: Pruebas para evaluar el costo computacional de los operadores

El cometido de estas pruebas es medir los tiempos de ejecución para cada uno de los operadores por separado. Para cada una de las pruebas definidas se ejecuta únicamente una ejecución independiente y se fija el número de generaciones en 1000,

la semilla inicial en 123456 y el tamaño de la población en 30 individuos. Se utilizan las mismas configuraciones que en la calibración de parámetros en cuanto a usar reglas laborales y al mecanismo de generación de soluciones iniciales.

7.2.3 Selección de semillas para las pruebas

Una vez calibrado el algoritmo se deben seleccionar los valores de semillas a utilizar en cada una de las ejecuciones independientes. Esta selección fue realizada en base a números primos para asegurar su independencia mutua. En base a esta decisión es que surgen los siguientes valores de semillas: 75077, 750119 y 750161.

La cantidad de semillas a utilizar en cada una de las configuraciones definidas debe ser mayor a uno debido a que al ser un algoritmo no determinista en cada ejecución se obtendrá un resultado distinto. A su vez teniendo en cuenta que el tiempo de ejecución de cada configuración es relativamente alto se debe optar por un número de semillas no demasiado grande. En virtud de estas dos cotas es que se entiende que tres es un número apropiado para la cantidad de semillas a utilizar.

7.2.4 Pruebas

Luego de fijar los valores de los parámetros a utilizar se presentarán los experimentos realizados para evaluar, analizar y luego comparar la solución implementada con otros resultados conocidos. Se definen, para evaluar los casos de estudio Héctor y COPSA, los siguientes escenarios los cuales modelan distintas realidades:

		Generación de Soluciones Iniciales	
		Modo 3	Modo 4
Reglas Laborales	NO	Escenario 1	Escenario 2
	SI	Escenario 3	Escenario 4

Tabla 7.3: Escenarios de pruebas

Como se puede observar, la diferencia principal entre los primeros dos escenarios y los últimos es la consideración o no de las reglas laborales. Los primeros dos escenarios, por tanto, se corresponden con el problema de ordenamiento de flota, mientras que los dos últimos resuelven el problema completo con ambas asignaciones. Cabe aclarar que las reglas utilizadas son las definidas en la Sección 6.1.4.

Con respecto al uso de los distintos modos de generación de soluciones iniciales, estas pruebas se presentan adicionalmente para aprovechar la flexibilidad que nos brindan el módulo GSI junto con el de Workflow aún sabiendo que el modo 3 le quita diversidad a la generación inicial. La falta de diversidad se debe a que, en este modo, todos los individuos tendrán servicios con recorridos de tan corta duración que en la mayor parte de los casos quedarán formados por un solo viaje, con lo cual las soluciones terminarán siendo muy similares entre sí. En cambio cuando se configura el modulo GSI para que genere individuos en base a microturnos de más larga duración, se obtienen individuos más diversos lo cual permite una mejor exploración del espacio de soluciones. Lo interesante del modo 3 es que, debido a sus características, se puede

observar con mayor precisión el trabajo del algoritmo genético sobre la evolución de la población de soluciones. Por otro lado, la motivación para hacer pruebas diferenciadas con y sin reglas laborales, es el hecho de que contamos con resultados de los proyectos antecesores con los cuales podemos realizar comparaciones para medir el desempeño de Marco Polo en el problema reducido de ordenamiento de flota.

Para cada caso de prueba se realizarán tres ejecuciones independientes con las distintas semillas definidas.

7.3 Resultados Obtenidos

7.3.1 Calibración

En las siguientes tablas (Tabla 7.4 y Tabla 7.5) se presentan los resultados obtenidos en cada una de las 72 pruebas definidas para realizar la calibración de parámetros, divididos en dos tandas según el tamaño de la población. Se resaltan las cinco configuraciones para las cuales se obtuvieron mejores costos (ejecuciones 18, 36, 46, 58 y 66):

Ejecución	población	cruza-miento	mutación exp.inter.	mutación unir turnos	mutación sep.turnos	costo ($)	tiempo (minutos)
1	30	0,2	0,1	0,2	0,1	24020	19,44
2	30	0,2	0,1	0,2	0,3	20661,7	23,93
3	30	0,2	0,1	0,5	0,1	21000	16,96
4	30	0,2	0,1	0,5	0,3	16605,8	19,27
5	30	0,2	0,1	0,9	0,1	20120	16,29
6	30	0,2	0,1	0,9	0,3	14406,7	17,17
7	30	0,2	0,3	0,2	0,1	18944,2	20,09
8	30	0,2	0,3	0,2	0,3	16965,8	24,5
9	30	0,2	0,3	0,5	0,1	19161,7	19,02
10	30	0,2	0,3	0,5	0,3	14901,7	21,87
11	30	0,2	0,3	0,9	0,1	17480	18,77
12	30	0,2	0,3	0,9	0,3	15761,7	18,17
13	30	0,5	0,1	0,2	0,1	19404,2	20,59
14	30	0,5	0,1	0,2	0,3	22511,7	29,82
15	30	0,5	0,1	0,5	0,1	17203,3	20,64
16	30	0,5	0,1	0,5	0,3	15368,3	22,41
17	30	0,5	0,1	0,9	0,1	17333,3	21,01
18	30	0,5	0,1	0,9	0,3	12589,2	22,26
19	30	0,5	0,3	0,2	0,1	20360	23,21
20	30	0,5	0,3	0,2	0,3	28741,7	35,98
21	30	0,5	0,3	0,5	0,1	19120	22,9
22	30	0,5	0,3	0,5	0,3	14152,5	26,24
23	30	0,5	0,3	0,9	0,1	18125	22,43
24	30	0,5	0,3	0,9	0,3	16120	22,93
25	30	0,9	0,1	0,2	0,1	18567,5	29,98
26	30	0,9	0,1	0,2	0,3	21974,2	39,49
27	30	0,9	0,1	0,5	0,1	18000	27,05
28	30	0,9	0,1	0,5	0,3	13709,2	29,13
29	30	0,9	0,1	0,9	0,1	18370	26,91
30	30	0,9	0,1	0,9	0,3	18000	28,34
31	30	0,9	0,3	0,2	0,1	19464,2	31,63
32	30	0,9	0,3	0,2	0,3	18829,2	41,52
33	30	0,9	0,3	0,5	0,1	19584,2	30,85
34	30	0,9	0,3	0,5	0,3	14483,3	36,76
35	30	0,9	0,3	0,9	0,1	19581,7	30
36	30	0,9	0,3	0,9	0,3	12328,3	30,3

Tabla 7.4: Resultados de la calibración para tamaño de población 30

Ejecución	población	cruza-miento	mutación exp.inter.	mutación unir turnos	mutación sep.turnos	costo ($)	tiempo (mm:ss)
37	50	0,2	0,1	0,2	0,1	18019,2	26,74
38	50	0,2	0,1	0,2	0,3	15383,3	32,05
39	50	0,2	0,1	0,5	0,1	18161,7	20,78
40	50	0,2	0,1	0,5	0,3	13284,2	24,99
41	50	0,2	0,1	0,9	0,1	19180	24,24
42	50	0,2	0,1	0,9	0,3	15829,2	24,52
43	50	0,2	0,3	0,2	0,1	19108,3	33,16
44	50	0,2	0,3	0,2	0,3	18519,2	46,19
45	50	0,2	0,3	0,5	0,1	18485	27,13
46	50	0,2	0,3	0,5	0,3	12630,8	36,02
47	50	0,2	0,3	0,9	0,1	18270,8	27,26
48	50	0,2	0,3	0,9	0,3	14547,5	28,01
49	50	0,5	0,1	0,2	0,1	15320,8	37,06
50	50	0,5	0,1	0,2	0,3	16246,7	41,09
51	50	0,5	0,1	0,5	0,1	16208,3	31,03
52	50	0,5	0,1	0,5	0,3	14170	33
53	50	0,5	0,1	0,9	0,1	19300	31,75
54	50	0,5	0,1	0,9	0,3	15170,8	33
55	50	0,5	0,3	0,2	0,1	19125	34,04
56	50	0,5	0,3	0,2	0,3	15821,7	40,83
57	50	0,5	0,3	0,5	0,1	18221,7	31,48
58	50	0,5	0,3	0,5	0,3	11974,2	35,05
59	50	0,5	0,3	0,9	0,1	18041,7	31,52
60	50	0,5	0,3	0,9	0,3	15354,2	33,97
61	50	0,9	0,1	0,2	0,1	18409,2	37,45
62	50	0,9	0,1	0,2	0,3	16838,3	49,41
63	50	0,9	0,1	0,5	0,1	16541,7	37,09
64	50	0,9	0,1	0,5	0,3	14292,5	41,06
65	50	0,9	0,1	0,9	0,1	16604,2	36,97
66	50	0,9	0,1	0,9	0,3	12755,8	39,54
67	50	0,9	0,3	0,2	0,1	18660	39,88
68	50	0,9	0,3	0,2	0,3	16821,7	50,76
69	50	0,9	0,3	0,5	0,1	16541,7	40,11
70	50	0,9	0,3	0,5	0,3	13124,2	41,16
71	50	0,9	0,3	0,9	0,1	16604,2	38,87
72	50	0,9	0,3	0,9	0,3	16505,8	39,35

Tabla 7.5: Resultados de la calibración para tamaño de población 50

Podemos observar que tres de los cinco mejores resultados se presentaron para el tamaño de población de 50 individuos pero que, a su vez, este tamaño influye negativamente en el costo computacional del algoritmo.

Con respecto a los operadores podemos ya deducir que la probabilidad del operador de Mutación Separar Turnos quedará fijada en 0,3 dado que es la configuración de dicho operador en los cinco mejores casos. Para el resto de los operadores sería conveniente analizar su comportamiento antes de sacar conclusiones.

El siguiente es el estudio del costo computacional de cada operador según el tiempo que emplea cada uno de ellos. Se ejecutaron los cuatro escenarios definidos en la sección 7.2.2 *Parámetros a calibrar* del presente capítulo:

Prueba de operadores	Operador evaluado	Tiempo (segundos)
1	Cruzamiento Recursivos de Subgrafos	323
2	Mutacion Expresos Intercambiables	220
3	Mutación Unir Turnos	180
4	Mutación Separar Turnos	175

Tabla 7.6: Resultados de las pruebas de costo computacional de los operadores

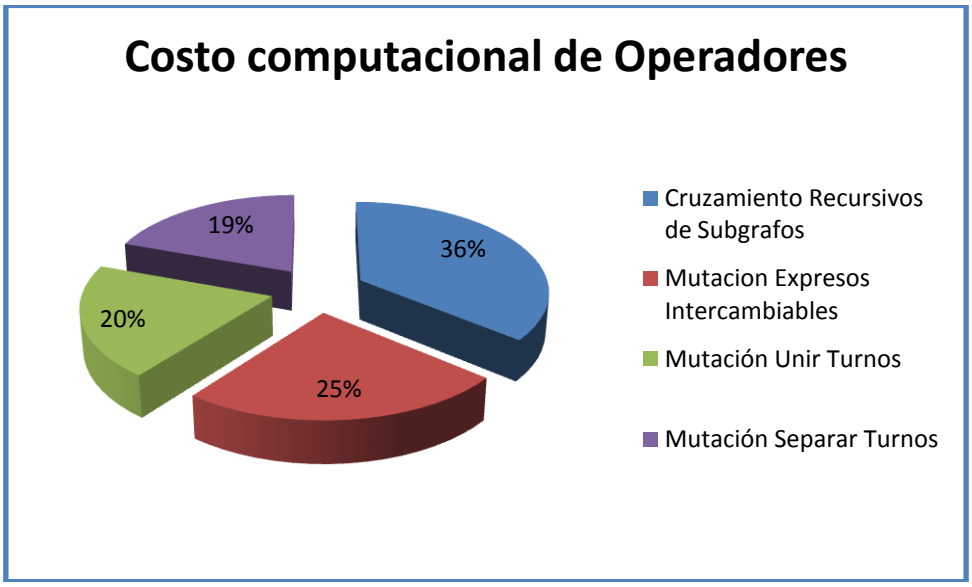

Figura 7.1: Gráfica de costo computacional de operadores

Como se puede ver en los resultados de las pruebas de costo computacional, el operador más costoso es el de Cruzamiento Recursivo de Subgrafos. De todas maneras, es importante destacar que se espera que dicho operador tenga una duración variable la cual dependerá de qué tan distintos o similares sean los progenitores a cruzar.

Con respecto a la Mutación Expresos Intercambiables es coherente que consuma más tiempo que los otros dos operadores de mutación puesto que, si se analiza al detalle su funcionamiento se puede ver que, esta termina realizando las dos tareas a la vez: en primer lugar separa los turnos correspondientes a los expresos a mutar y luego los une de acuerdo a su nueva configuración. De todas maneras la ejecución de la Mutación Expresos Intercambiables queda condicionada a que en el individuo a mutar existan expresos y además estos sean intercambiables.

En cuanto a los operadores de mutación Unir Turnos y Separar Turnos se observa que el tiempo empleado en ambos casos es muy parecido lo cual se justifica dado que su operativa es similar.

Conociendo ahora la diferencia entre los tiempos empleados por cada operador podemos analizar con más detalle las mejores ejecuciones obtenidas. A continuación se reúnen las mejores cinco configuraciones ordenando el costo de manera decreciente:

Ejecución	población	cruza-miento	mutación exp.inter.	mutación unir turnos	mutación sep.turnos	costo ($)	tiempo (mm:ss)
58	50	0,5	0,3	0,5	0,3	11974,2	35,05
36	30	0,9	0,3	0,9	0,3	12328,3	30,3
18	30	0,5	0,1	0,9	0,3	12589,2	22,26
46	50	0,2	0,3	0,5	0,3	12630,8	36,02
66	50	0,9	0,1	0,9	0,3	12755,8	39,54

Tabla 7.7: Mejores configuraciones

Aunque las ejecuciones de mejor costo con una población de 30 individuos se ubican en el segundo y tercer puesto de la Tabla 7.7, sabemos que una población de mayor tamaño aporta más diversidad al algoritmo genético.

Con respecto a la probabilidad de cruzamiento, como era de esperar, predominan las probabilidades más altas. Considerando que, según el análisis presentado previamente, es el operador más costoso decidimos que la probabilidad moderada de la primera ejecución, también para este parámetro, es la más indicada.

Observando la ejecución con mejor costo obtenida vemos que su duración se encuentra exactamente en medio de los tiempos empleados por las cinco mejores ejecuciones, en donde vemos que dos de ellas se encuentran por debajo de su tiempo y las otras dos por encima.

En definitiva las características deseables más destacadas son que exista diversidad en la población y que se obtengan buenos costos tanto en términos del fitness como computacionales, con lo cual concluimos que la mejor configuración para los parámetros del algoritmo genético es la correspondiente a la ejecución 58 la cual se presenta en la Tabla 7.8:

Parámetro	Valor
Tamaño de la población	50
p(Cruzamiento Recursivos de Subgrafos)	0,5
p(Mutacion Expresos Intercambiables)	0,3
p(Mutación Unir Turnos)	0,5
p(Mutación Separar Turnos)	0,3

Tabla 7.8: Mejor configuración

Adicionalmente, en la Tabla 7.9, se registran los datos estadísticos de costo promedio y su desviación estándar así como también el tiempo de ejecución promedio:

	Valor
Costo promedio	$17278,1
Desviación Estándar	$2866,2
Tiempo Promedio	30,06 minutos

Tabla 7.9: Datos estadísticos

7.3.2 Caso de estudio Héctor

En esta sección se presentan los resultados obtenidos de la ejecución del plan de pruebas para el caso de estudio Héctor. Se observa que los escenarios de prueba definidos pueden ser categorizados en dos grandes grupos si se toma en cuenta la aplicación o no de las restricciones vinculadas a las reglas laborales. Este factor sin lugar a dudas es el que define el tipo de problema que se está atacando.

En primer lugar veremos los resultados de los escenarios 1 y 2, los cuales no tienen en cuenta las reglas laborales, por lo que se trata del problema simplificado que solo realiza el ordenamiento de la flota. Acompañando los resultados se muestra gráficamente, en cada caso, la evolución de las distintas magnitudes de interés para la ejecución con mejor costo:

Semilla	Costo	Tiempo (minutos)	#km. expresos	#Turnos	#Vehículos
750077	12375	15,39	0	7	6
750119	11677	11,96	40	7	6
750161	11208	12,33	0	6	6

Tabla 7.10: Resultados Escenario 1 – Caso Héctor

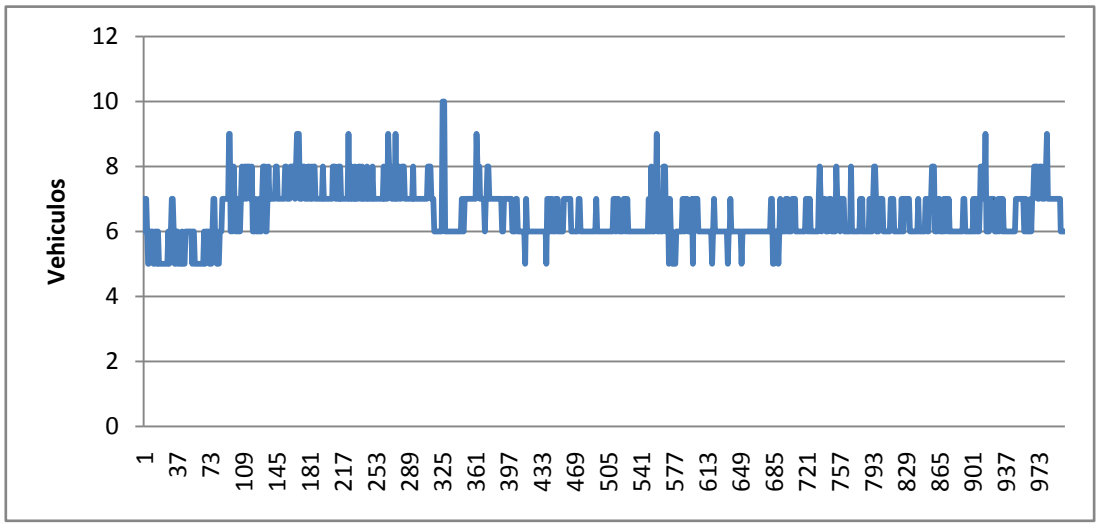

Figura 7.2: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750161)

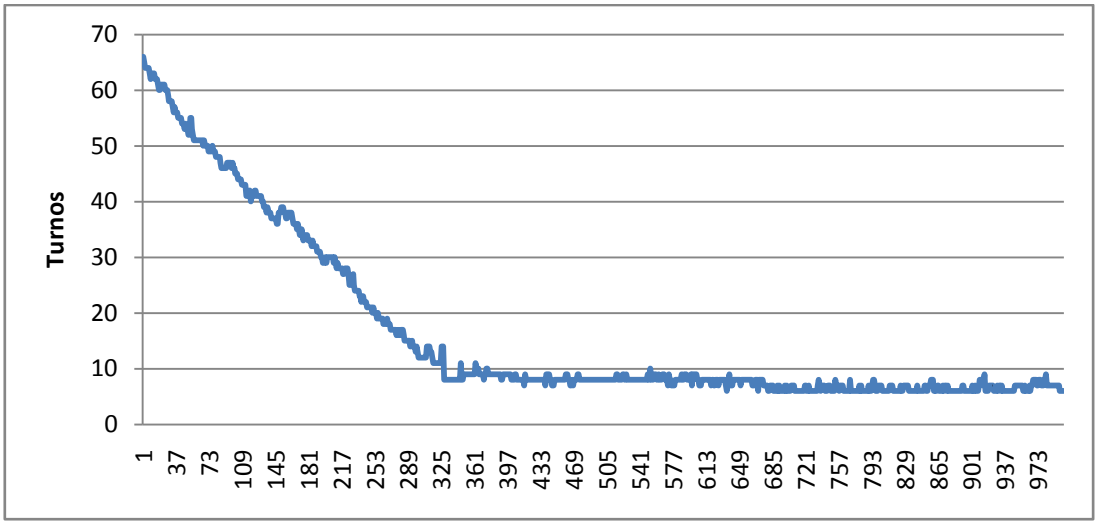

Figura 7.3: Evolución de los Turnos a lo largo de las iteraciones (semilla 750161)

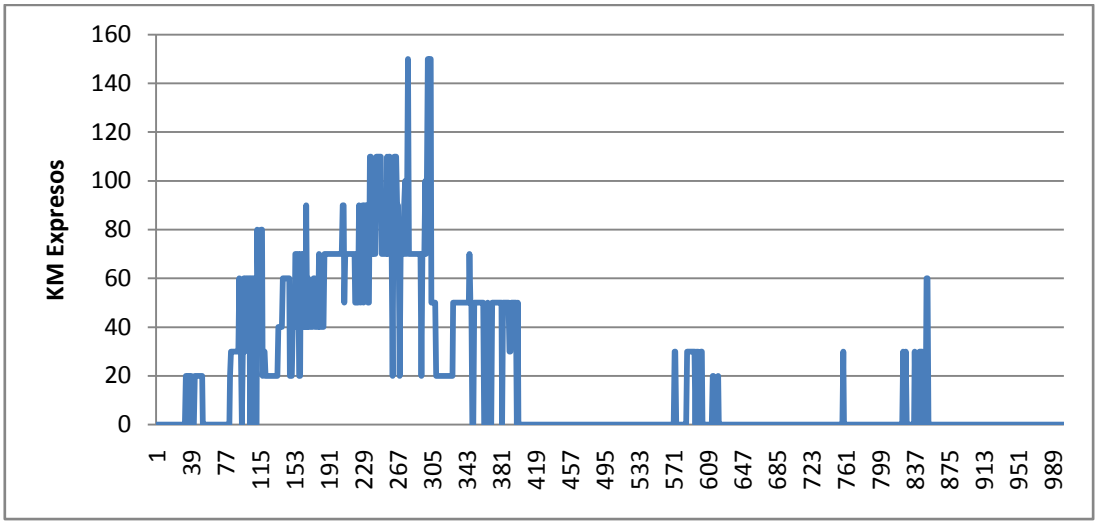

Figura 7.4: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750161)

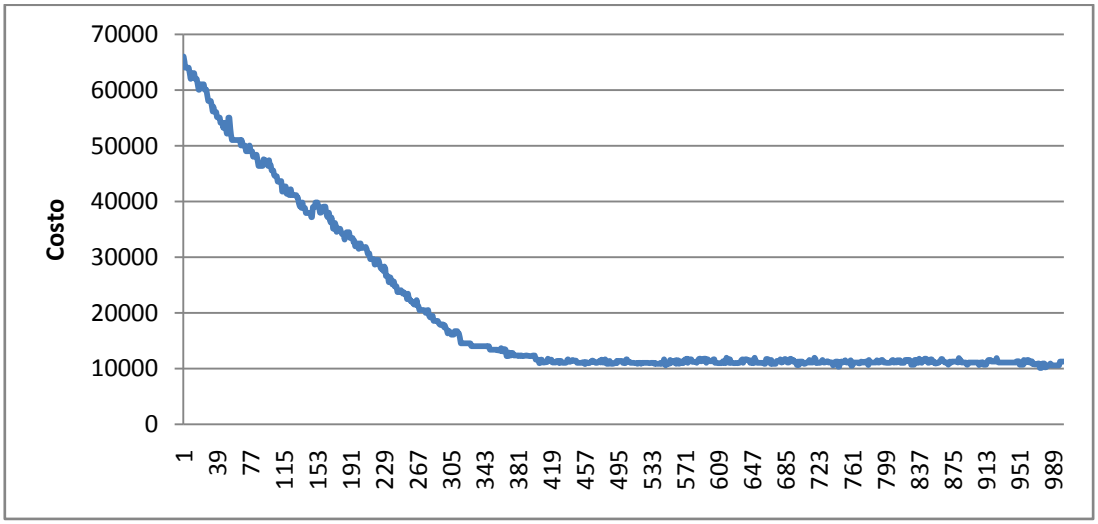

Figura 7.5: Evolución del Costo a lo largo de las iteraciones (semilla 750161)

Semilla	Costo	Tiempo (minutos)	#km. expresos	#Turnos	#Vehículos
750077	10270	10,87	0	8	6
750119	10270	11,04	0	9	6
750161	10125	10,93	0	9	6

Tabla 7.11: Resultados Escenario 2 – Caso Héctor

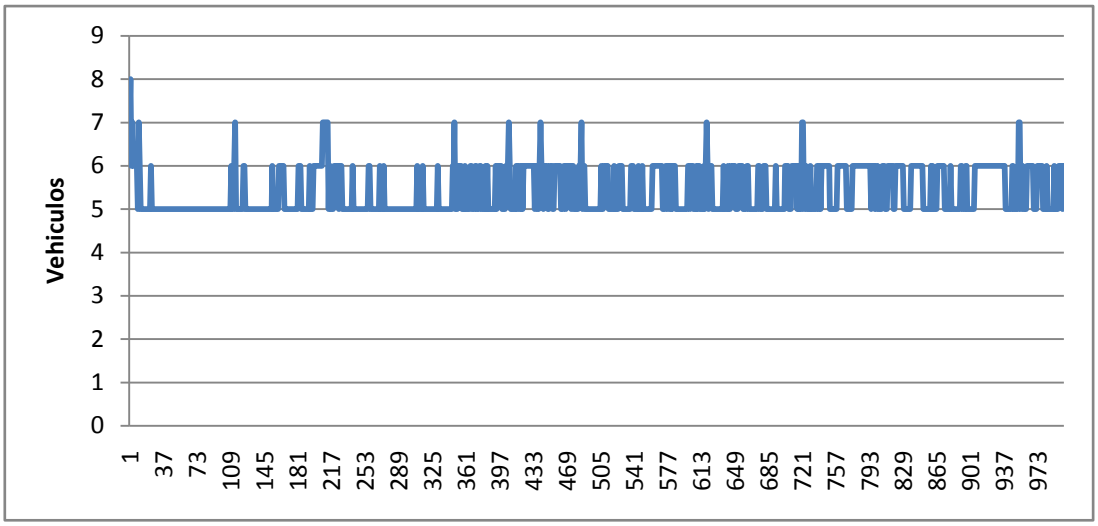

Figura 7.6: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750161)

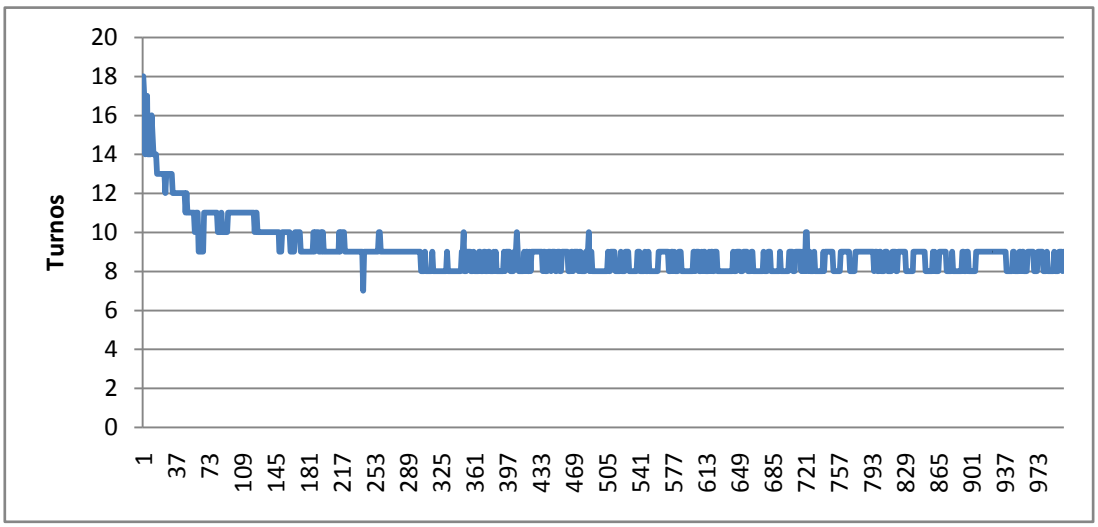

Figura 7.7: E volución de los Turnos a lo largo de las iteraciones (semilla 750161)

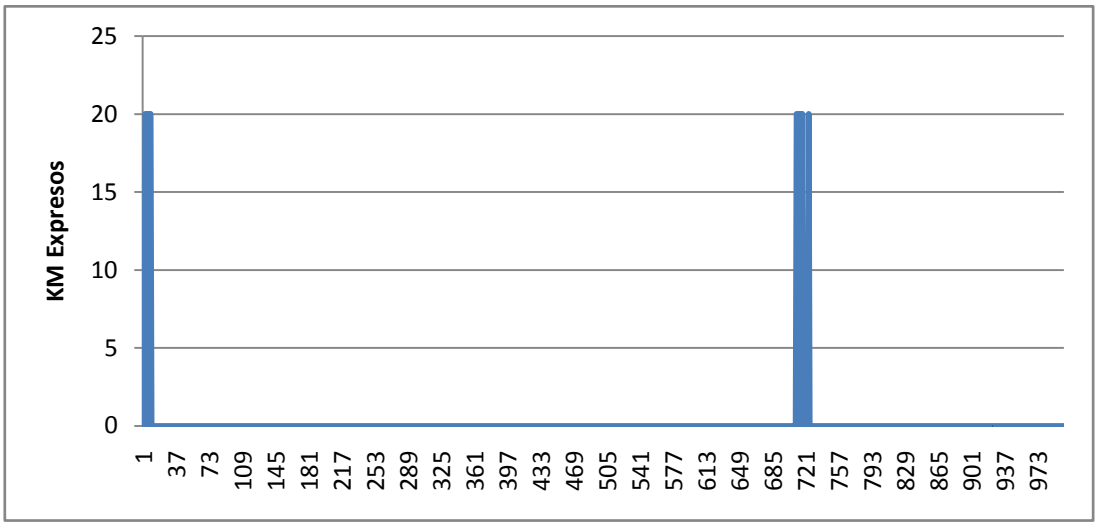

Figura 7.8: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750161)

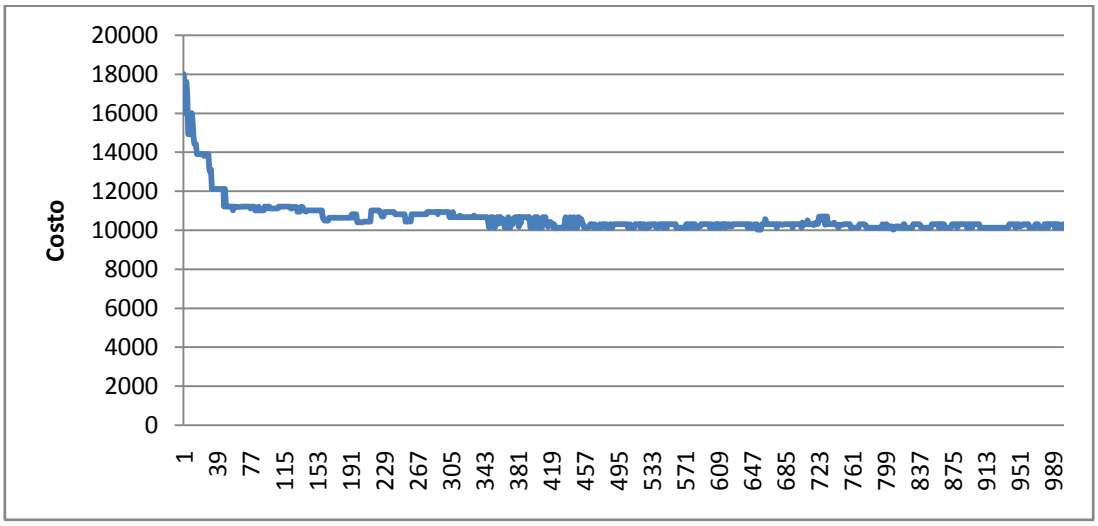

Figura 7.9: Evolución del Costo a lo largo de las iteraciones (semilla 750161)

Estos dos escenarios de prueba hacen especial énfasis en la resolución del problema de ordenamiento de la flota de vehículos dado que no se tienen restricciones explícitas sobre los turnos. Cabe aclarar que, si bien no se tienen en cuenta las reglas laborales, la función que mide la aptitud de los individuos de la población considera el costo de los turnos presentes en el individuo y ello incide sobre la evolución en la búsqueda de la mejor solución. En este contexto resulta entonces esperable alcanzar una cota mínima en la cantidad de turnos.

Para este caso de estudio se obtuvieron, en 5 de 6 ejecuciones, 0 kilómetros expresos mientras que en la restante ejecución se obtuvieron 40. Para los casos en que se obtuvieron 0 kilómetros expresos se alcanzó el valor óptimo del factor.

Finalmente veremos los escenarios más importantes que atacan el problema integrado de ordenamiento de flota y tripulación. Se trata de los escenarios 2 y 3 definidos en el plan de pruebas:

Semilla	Costo	Tiempo (minutos)	#km. expresos	#Turnos	#Vehículos
750077	11740	10,2	40	11	5
750119	13920	10,36	70	12	6
750161	12573	11,43	40	10	6

Tabla 7.12: Resultados Escenario 1 – Caso Héctor

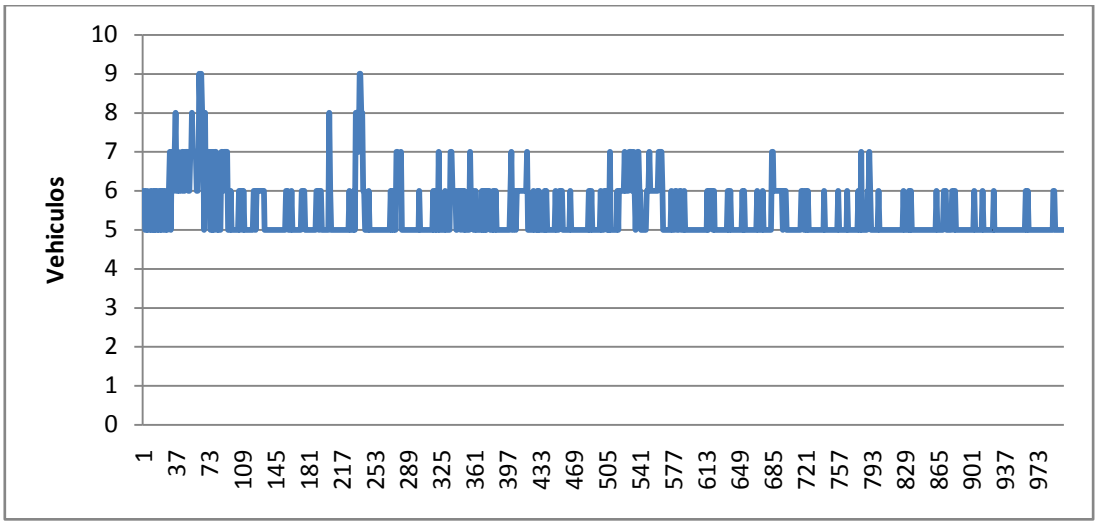

Figura 7.10: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750077)

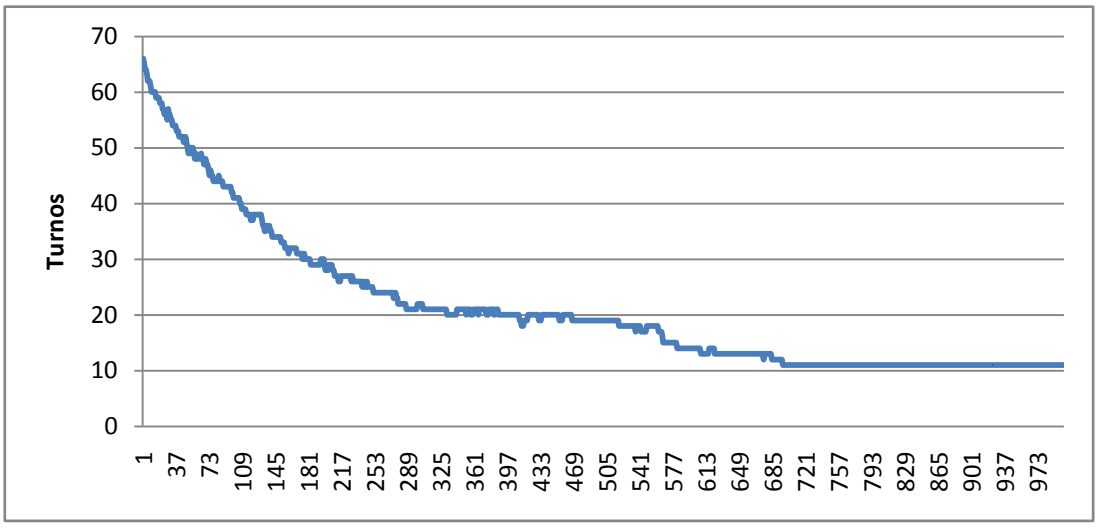

Figura 7.11: Evolución de los Turnos a lo largo de las iteraciones (semilla 750077)

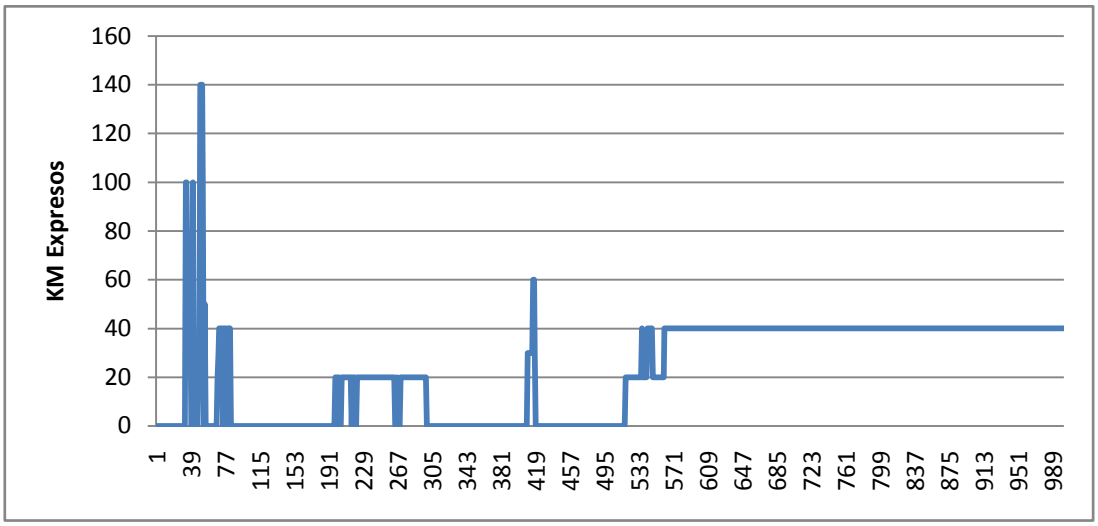

Figura 7.12: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750077)

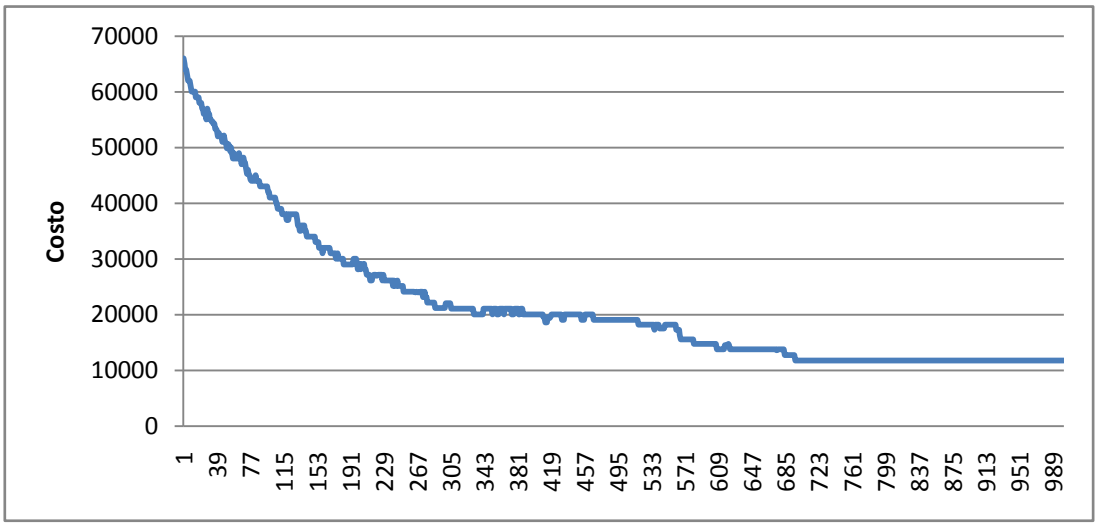

Figura 7.13: Evolución del Costo a lo largo de las iteraciones (semilla 750077)

Semilla	Costo	Tiempo (minutos)	#km. expresos	#Turnos	#Vehículos
750077	15120	10,75	20	15	6
750119	14781	12,01	40	14	8
750161	13526	12,2	60	13	6

Tabla 7.13: Resultados Escenario 4 – Caso Héctor

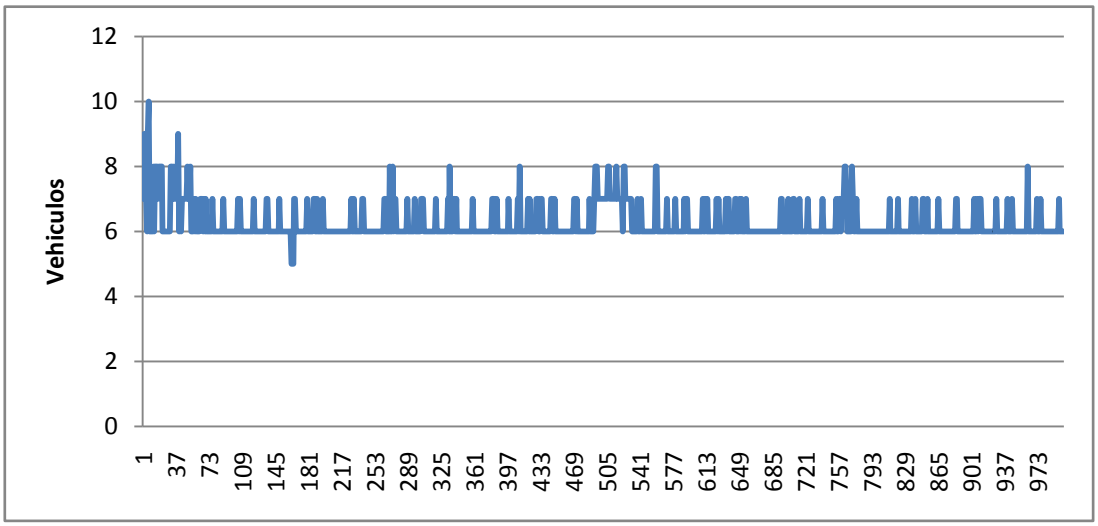

Figura 7.14: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750161)

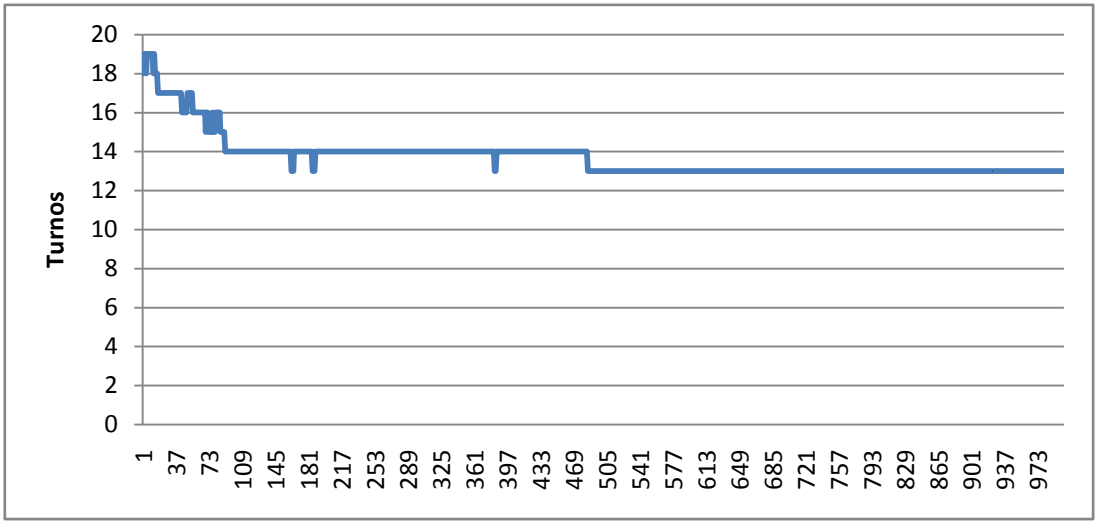

Figura 7.15: Evolución de los Turnos a lo largo de las iteraciones (semilla 750161)

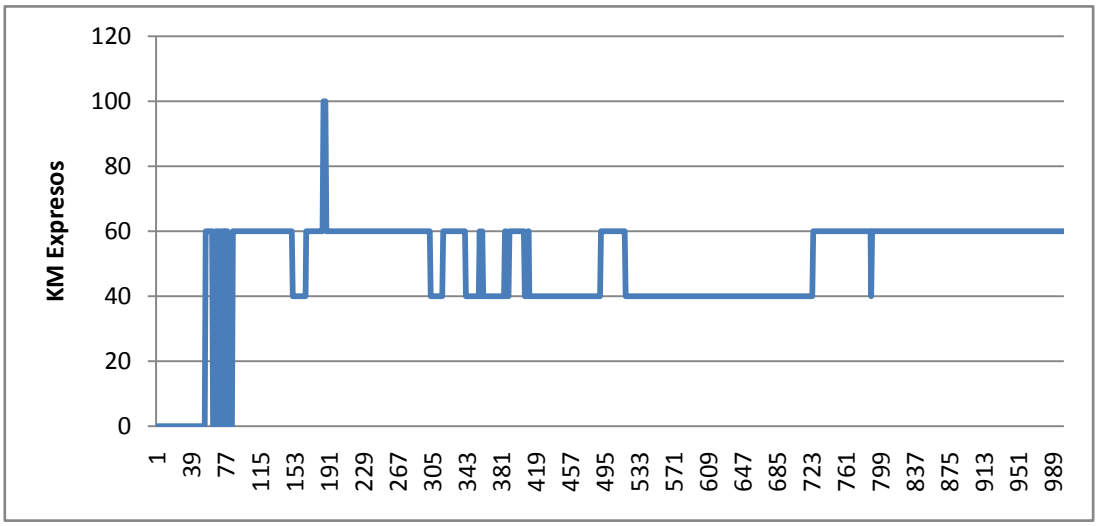

Figura 7.16: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750161)

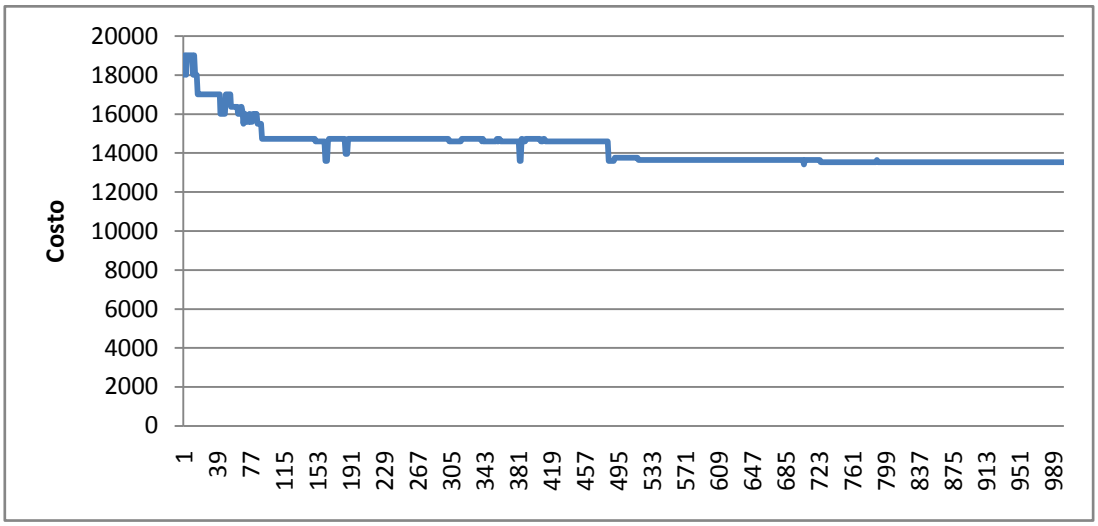

Figura 7.17: Evolución del Costo a lo largo de las iteraciones (semilla 750161)

Dado que aquí sí se ataca la resolución del problema de ordenamiento de la flota de manera integrada con la asignación de turnos los principales factores a analizar son los valores obtenidos de kilómetros expresos y cantidad de turnos.

Las pruebas efectuadas arrojaron valores de kilómetros expresos entre 20 y 70. En ninguno de los casos se llego al óptimo para este factor (0 kilómetros), pero en 4 de las 6 ejecuciones se llegó al peor caso obtenido entre los escenarios 1 y 2 (40 kilómetros), mientras que en las restantes dos ejecuciones se obtuvo un incremento del 75% (70 kilómetros) y 50 % (60 kilómetros) en los kilómetros expresos obtenidos respecto al mencionado caso.

Analizando el factor turnos, Marco Polo obtuvo en las anteriores seis ejecuciones valores entre 10 y 15. La gráfica de evolución de turnos muestra como con aproximadamente 500 iteraciones se alcanza el valor final para este factor.

7.3.2.1 Conclusiones generales: Caso de estudio Héctor

Para el caso de estudio Héctor, tanto al resolver el problema simplificado de ordenamiento de flota como al atacar el problema integrado con asignación de tripulación, podemos apreciar poca variación en los números en cuanto a los distintos métodos de generación de soluciones iniciales. Este comportamiento puede deberse a la magnitud del caso de estudio, tamaño para el cual no es necesario tener tanta diversidad al comienzo de la ejecución. Igualmente se observan resultados ligeramente mejores con el modo 3 de generación de soluciones iniciales, el cual se corresponde con el caso de un viaje por microturno.

Adicionalmente, observando las gráficas de evolución de los distintos factores, se observa que los valores finales se obtienen tempranamente, no siendo necesario aparentemente realizar una cantidad tan elevada de iteraciones. Este comportamiento también podría deberse al reducido tamaño del caso de estudio. Experimentalmente se concluye que para el presente caso de estudio, el criterio de parada debe aproximarse a la mitad de las generaciones utilizadas durante las pruebas realizadas, es decir que, es suficiente con que se realicen unas 500 generaciones

aproximadamente para alcanzar la convergencia del algoritmo. Esto implica una reducción del 50% del tiempo de ejecución.

Constatamos que se cumple el hecho de que los dos primeros escenarios fijan una cota inferior en cuanto a cantidad de turnos y luego los escenarios con reglas laborales arrojan soluciones con cantidades próximas a dicha cota.

7.3.3 Caso de estudio COPSA

En el caso de estudio COPSA se decide que, para los escenarios 1 y 2 definidos en el plan de pruebas (sección 7.2.4), se realizará una única ejecución independiente para cada uno de ellos. Esto se justifica en base al elevado tiempo empleado para la ejecución del algoritmo con este caso de estudio, siendo que estas dos pruebas se corresponden con el caso de ordenamiento de flota sin reglas laborales, el cual no es el verdadero problema que se intenta resolver en Marco Polo.

Semilla	Costo	Tiempo (horas)	#km. expresos	#Turnos	#Vehículos
750161	902293	10,07	2433	874	132

Tabla 7.14: Resultados Escenario 1 – Caso COPSA

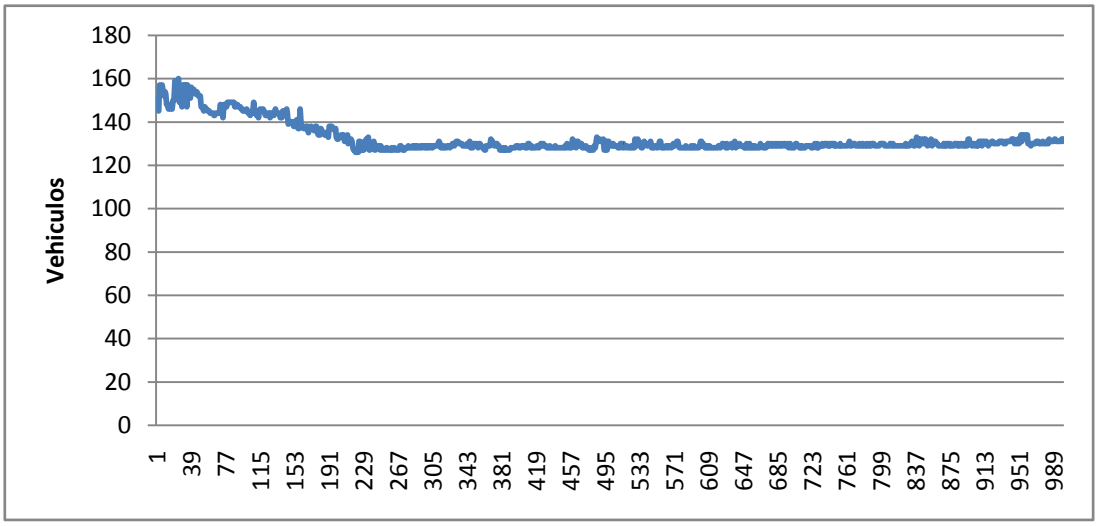

Figura 7.18: Evolución de los Vehículos a lo largo de las iteraciones

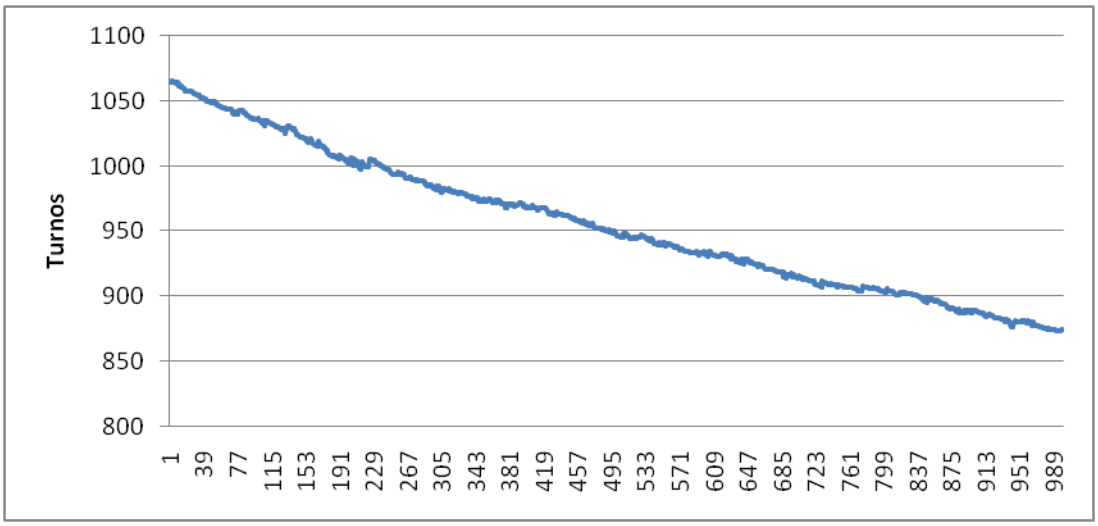

Figura 7.19: Evolución de los Turnos a lo largo de las iteraciones

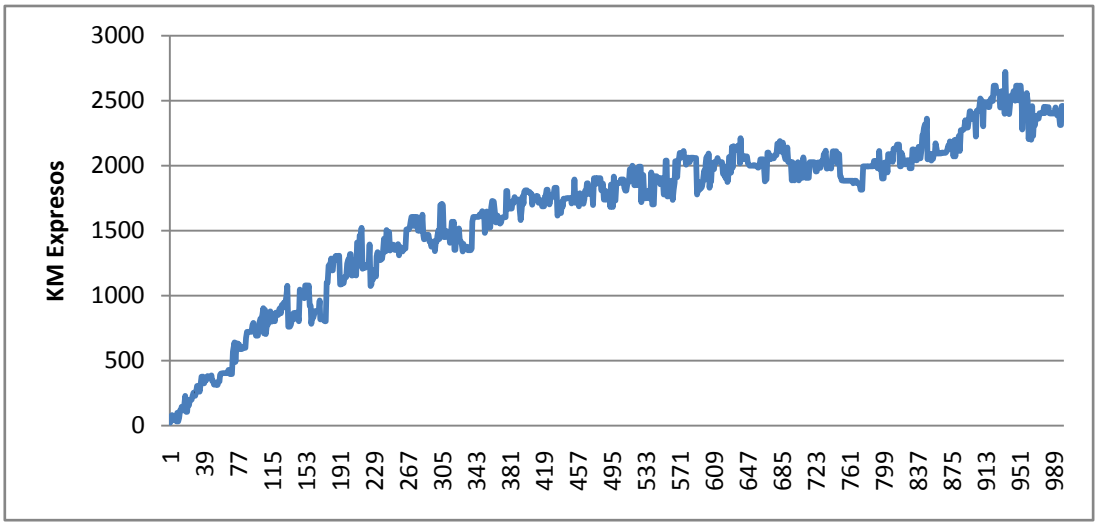

Figura 7.20: Evolución de los Km. Expresos a lo largo de las iteraciones

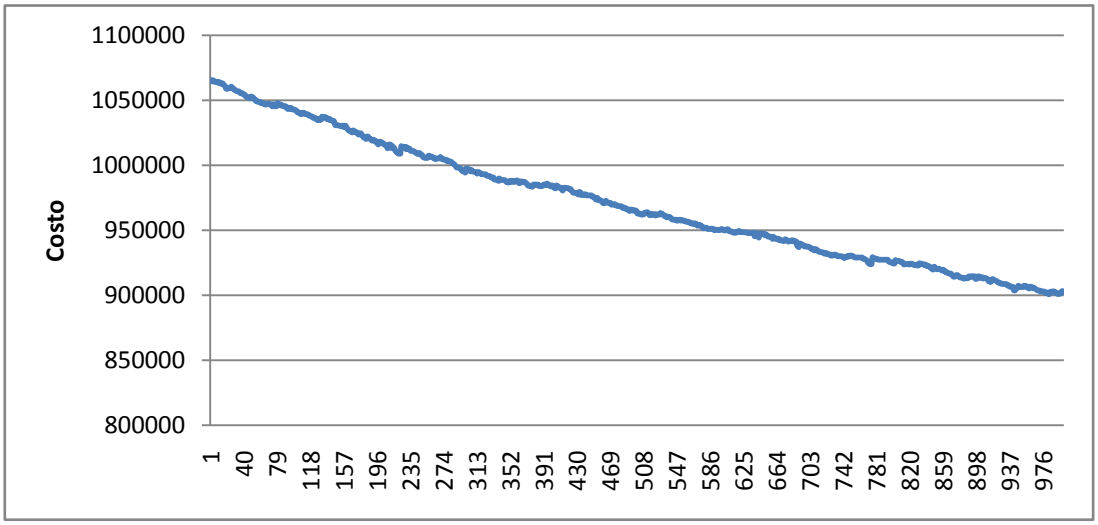

Figura 7.21: Evolución del Costo a lo largo de las iteraciones

Semilla	Costo	Tiempo (horas)	#km. expresos	#Turnos	#Vehículos
750161	481201	11,21	5551	385	157

Tabla 7.15: Resultados Escenario 2 – Caso COPSA

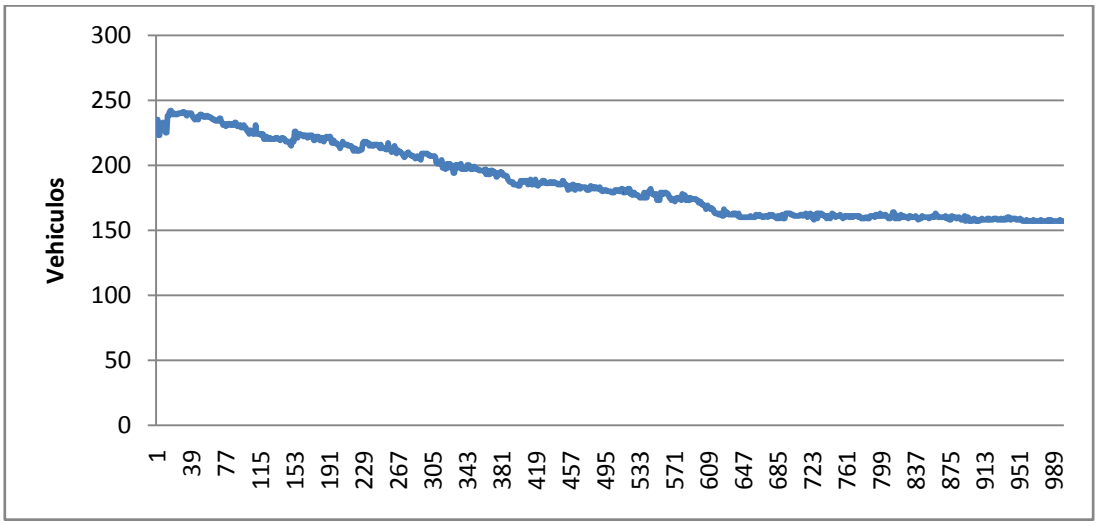

Figura 7.22: Evolución de los Vehículos a lo largo de las iteraciones

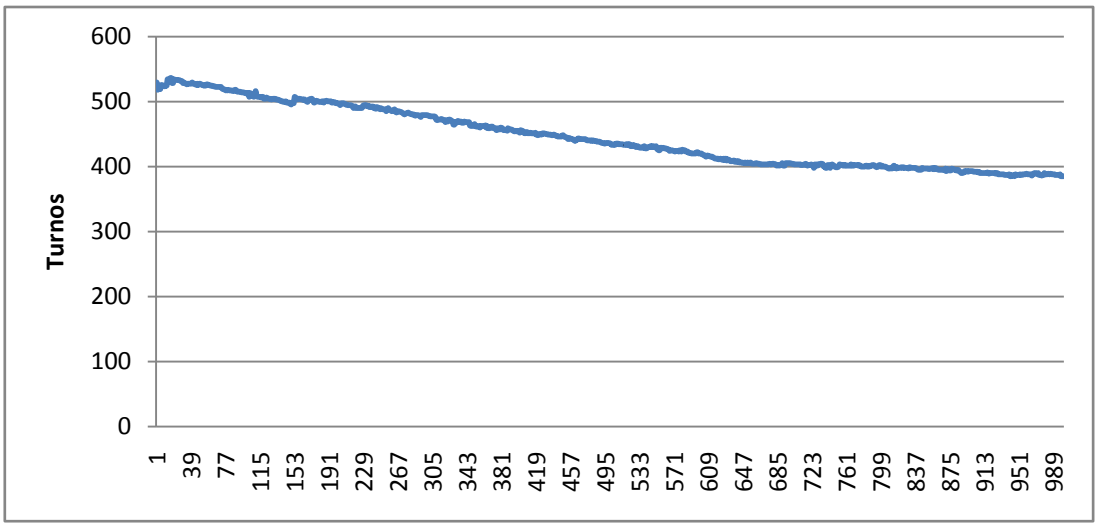

Figura 7.23: Evolución de los Turnos a lo largo de las iteraciones

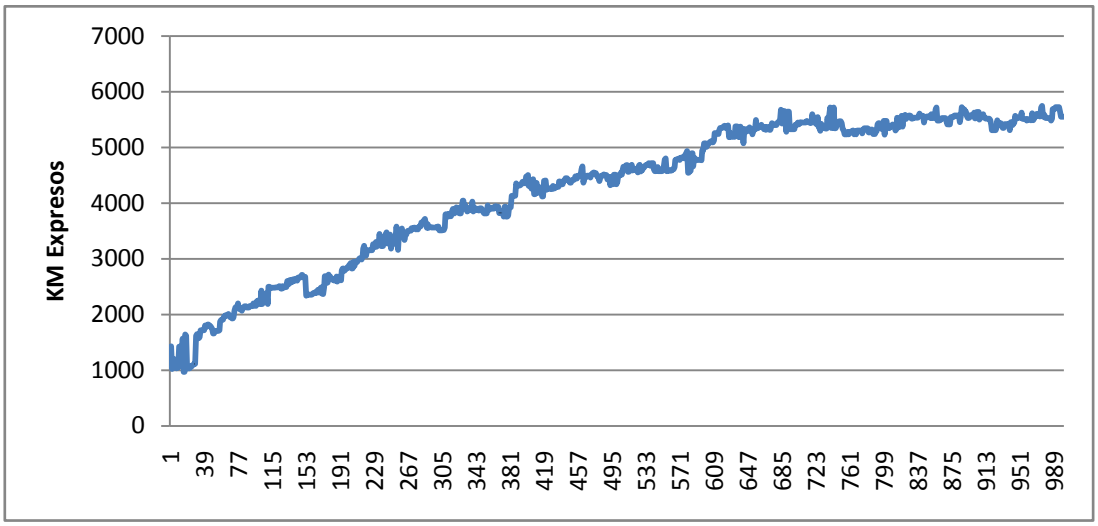

Figura 7.24: Evolución de los Km. Expresos a lo largo de las iteraciones

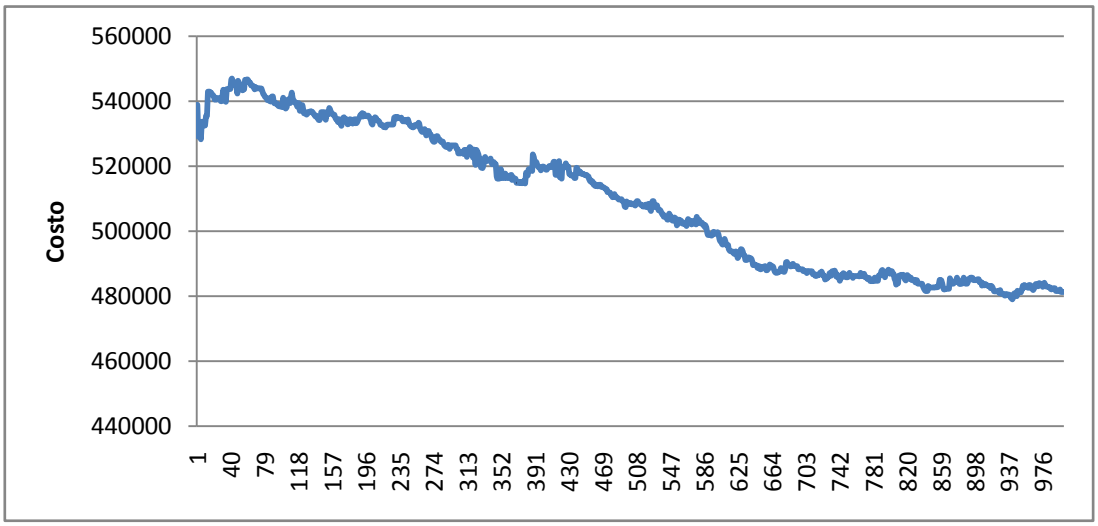

Figura 7.25: Evolución del Costo a lo largo de las iteraciones

Al igual que en las ejecuciones para el caso de estudio anterior, en este contexto sin reglas laborales, se espera alcanzar una cota mínima en la cantidad de turnos. Observando los resultados se establece como cota mínima el valor de 385 turnos.

Para el caso del escenario 1 se observa en la gráfica de evolución del costo a lo largo de las iteraciones la no convergencia del algoritmo. Esto se ve reflejado además en los valores de los distintos factores de interés. Para el escenario 2, en cambio, los resultados observados son coherentes con el comportamiento esperado del algoritmo luego de ver su desempeño en un caso de estudio de menor porte.

Para los escenarios 3 y 4 se presentan a continuación los resultados de las tres corridas independientes con las semillas seleccionadas:

Semilla	Costo	Tiempo (horas)	#km. expresos	#Turnos	#Vehículos
750077	911319	9,94	2699	895	127
750119	906672	9,43	2612	891	123
750161	918736	9,93	1956	907	123

Tabla 7.16: Resultados Escenario 3 – Caso COPSA

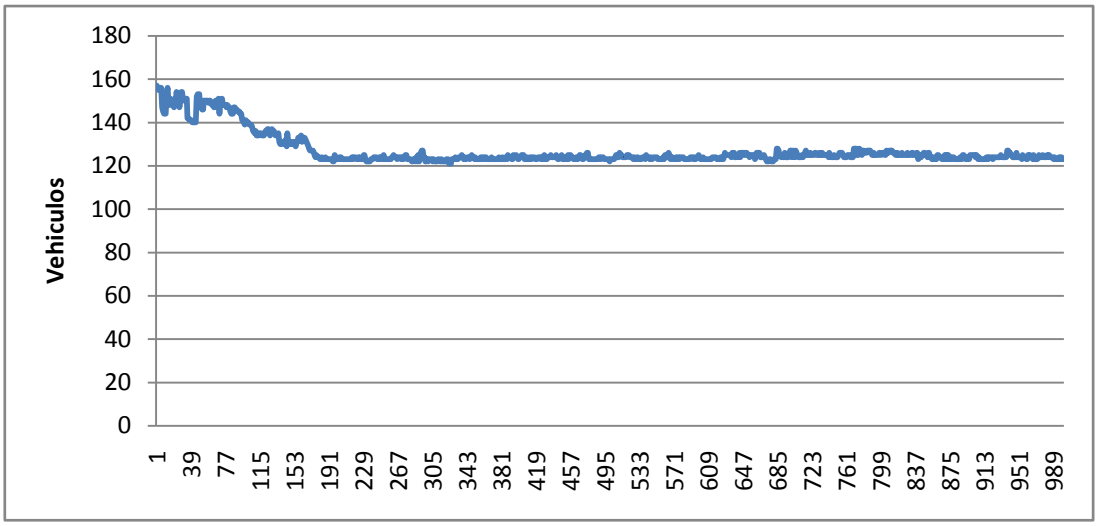

Figura 7.26: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750119)

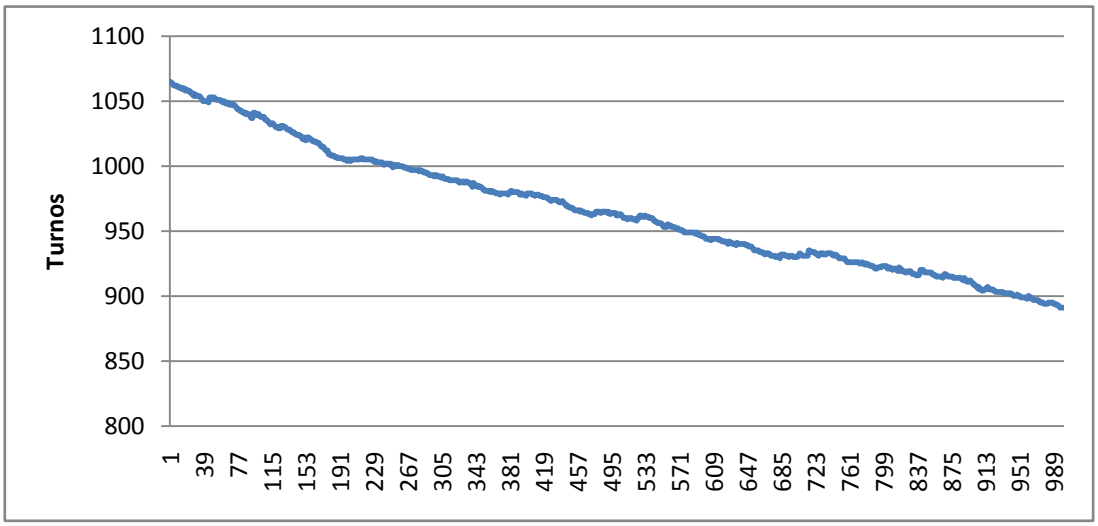

Figura 7.27: Evolución de los Turnos a lo largo de las iteraciones (semilla 750119)

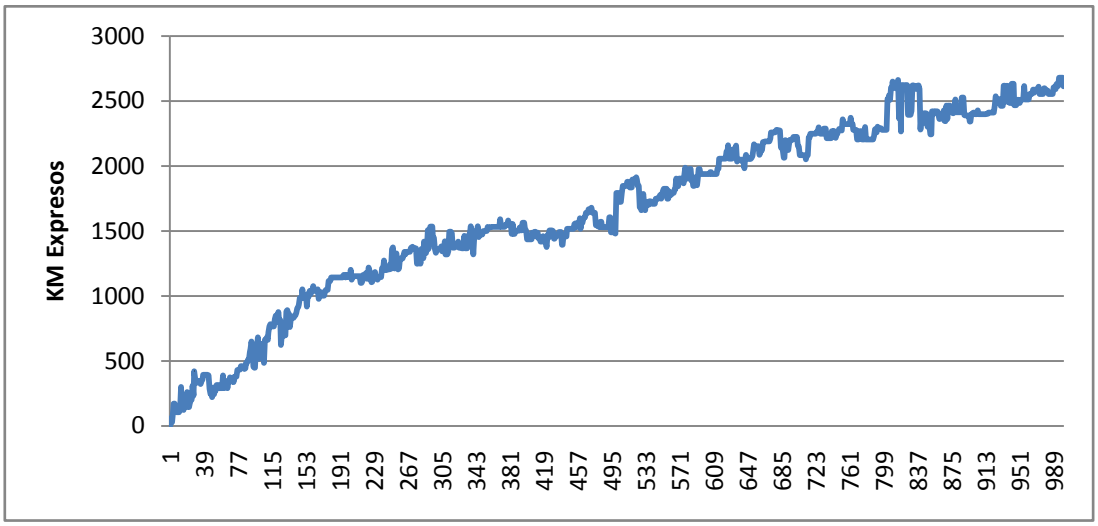

Figura 7.28: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750119)

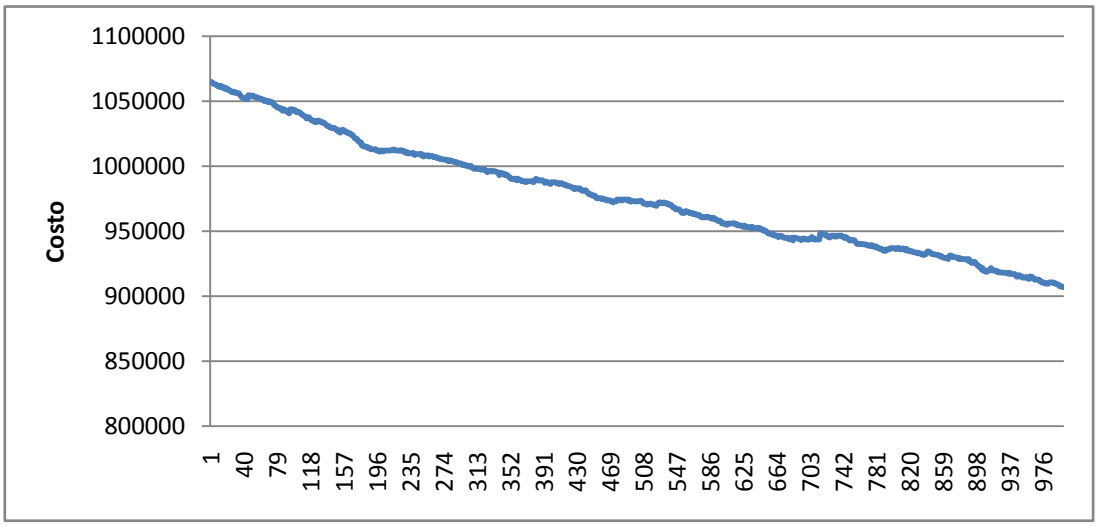

Figura 7.29: Evolución del Costo a lo largo de las iteraciones (semilla 750119)

Semilla	Costo	Tiempo (horas)	#km. expresos	#Turnos	#Vehículos
750077	511036	13,19	4202	479	147
750119	491372	11,48	4828	459	151
750161	488436	10,53	4820	453	150

Tabla 7.17: Resultados Escenario 4 – Caso COPSA

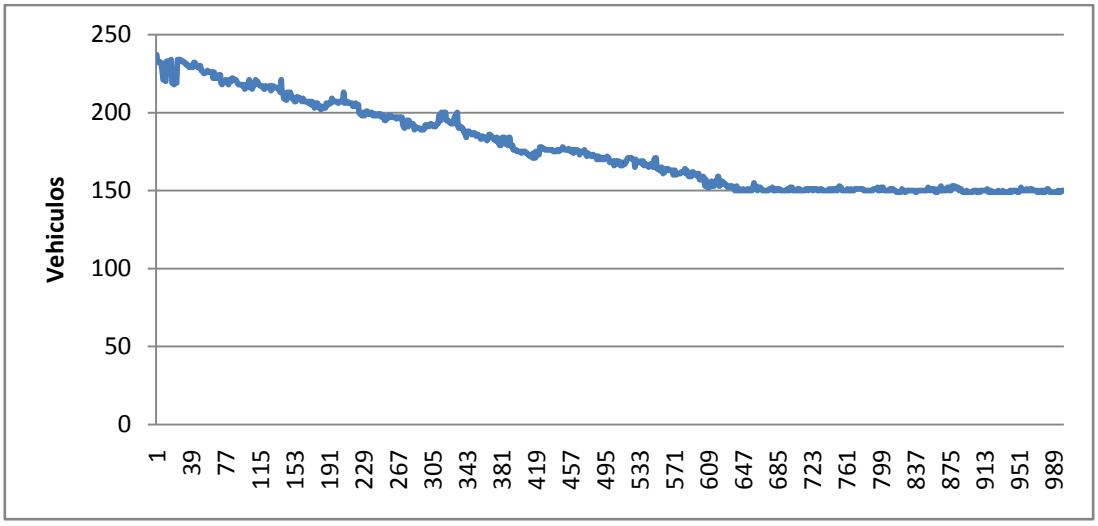

Figura 7.30: Evolución de los Vehículos a lo largo de las iteraciones (semilla 750161)

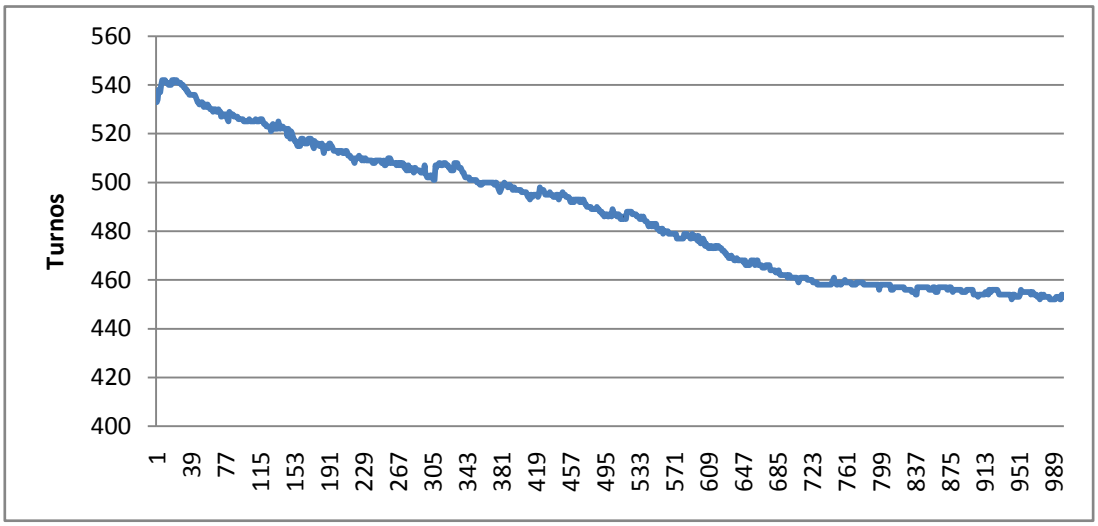

Figura 7.31: Evolución de los Turnos a lo largo de las iteraciones (semilla 750161)

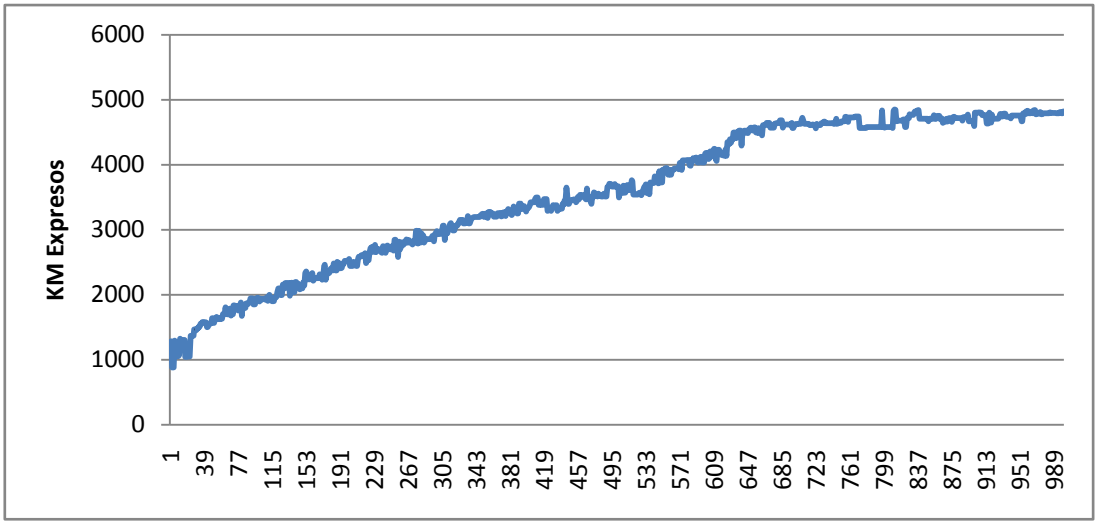

Figura 7.32: Evolución de los Km. Expresos a lo largo de las iteraciones (semilla 750161)

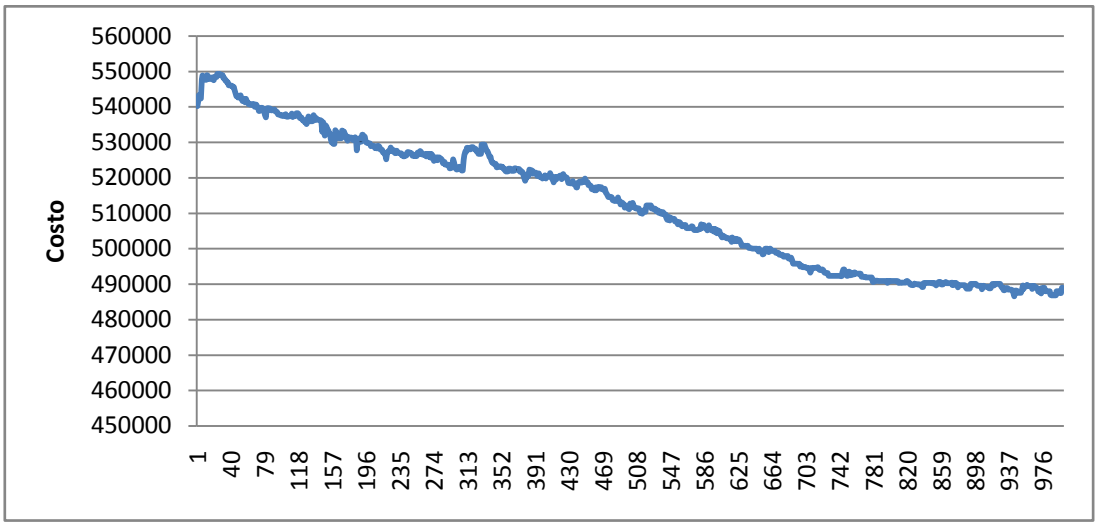

Figura 7.33: Evolución del Costo a lo largo de las iteraciones (semilla 750161)

Para el caso del escenario 3 se observa en la gráfica de evolución del costo la no convergencia del algoritmo al igual que sucedió con el escenario 1. La característica compartida por estos dos escenarios es el modo de generación de soluciones iniciales. Por lo tanto, en cuanto a las pruebas del caso COPSA con reglas laborales, se analizarán únicamente los datos obtenidos en el escenario 4.

Las pruebas efectuadas para el escenario 4 arrojaron valores de kilómetros expresos entre 4202 y 4828. En ninguno de los casos se llego al óptimo para este factor (0 kilómetros). Analizando el factor turnos para este mismo escenario, Marco Polo obtuvo valores entre 453 y 479. Las gráficas de evolución de los distintos factores muestran que la cantidad de iteraciones parece ser suficiente para alcanzar la convergencia en este caso.

7.3.3.1 Conclusiones generales: Caso de estudio COPSA

Tras los resultados obtenidos en los escenarios 1 y 3, en los cuales las soluciones iniciales resultaron ser extremadamente malas dado que los turnos presentan una duración muy reducida, se deduce observando las gráficas que la cantidad de iteraciones con la cual se han ejecutado las pruebas resulta insuficiente para lograr la convergencia hacia un resultado esperado. Si bien en estos escenarios se está lejos de los resultados deseados (sección 1.4 *Resultados esperados*), los datos obtenidos resultan consistentes dada la calidad de las soluciones iniciales, la complejidad del caso de estudio y la cantidad de evoluciones empleadas. Realizado este análisis se entiende que analizar los valores obtenidos en los escenarios 1 y 3 en función de otros valores carece de sentido dado que, para este caso de estudio, el modo de generación de soluciones iniciales resultó incidir notablemente en los resultados finales, con lo cual nos centraremos en los resultados obtenidos para los escenarios 2 y 4. Constatamos, también para este caso de estudio, que se cumple el hecho de que los dos primeros escenarios fijan una cota inferior en cuanto a cantidad de turnos.

En la totalidad de los escenarios para el caso de estudio de COPSA se observa una relación casi inversamente proporcional entre los kilómetros expresos y la cantidad de turnos.

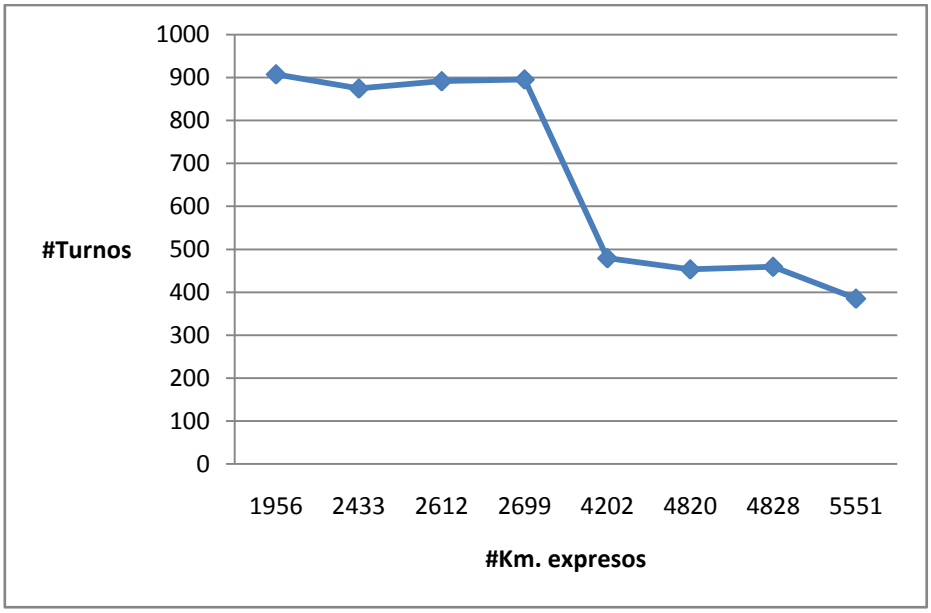

Figura 7.34: Relación entre #turnos y #km.expresos

7.4 Comparaciones

En esta sección los resultados obtenidos por Marco Polo serán comparados con tres sistemas o técnicas que fueron utilizadas para resolver el problema de ordenamiento de flota y asignación de tripulación.

Las comparaciones se harán con los resultados obtenidos por el proyecto PG42, proyecto predecesor de Marco Polo, con los resultados obtenidos de forma manual por los expertos de la empresa COPSA y con la solución obtenida por el grupo de investigación operativa en el marco del proyecto "Ordenamiento de Vehículos 2008-v1", en adelante el "Proyecto IO". Para las comparaciones con el PG42 contamos con datos de escenarios con y sin reglas laborales y ejecuciones sobre los casos de estudio Héctor y COPSA. Por su parte, la solución obtenida por el experto de la empresa, es con reglas laborales y no se cuenta con la resolución manual del caso Héctor con lo cual se podrá comparar el desempeño solo contra el caso COPSA con reglas laborales. Con respecto a la solución desarrollada por el grupo de IO, esta fue ejecutada sobre un escenario con reglas laborales pero no realiza un cubrimiento total de los viajes activos. Aún así se han podido obtener buenos resultados con un 93,9% de cobertura con lo cual consideramos que esta comparación es útil. Se cuenta con resultados obtenidos para ambos casos de estudio.

7.4.1 Comparaciones – Caso Héctor

	#Viajes	#km. expresos	#Turnos	#Vehículos
PG42	66	180	10	5
Marco Polo	66	0	9	6

Tabla 7.18: Escenario Sin Reglas Laborales – Caso Héctor

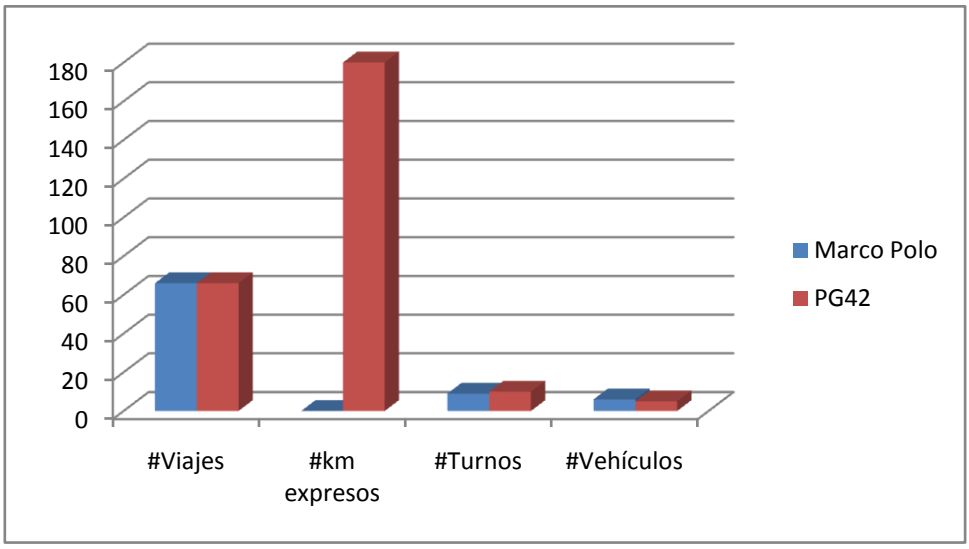

Figura 7.35: Gráfica comparativa - Escenario sin reglas laborales

En el escenario sin reglas laborales se observa que el desempeño de Marco Polo es muy bueno con respecto al dato de referencia utilizado. En términos porcentuales se obtuvo una disminución de los kilómetros expresos en un 100% dado que se obtuvo el valor óptimo para este factor. En relación a la cantidad de turnos obtenidos se observa también una reducción, esta vez, de un 10% con respecto al valor de referencia. Los turnos obtenidos en Marco Polo se distribuyen en 6 vehículos mientras que en el PG42 se utilizan 5 vehículos. Esta pequeña diferencia puede deberse a que en dicho proyecto se incluye en la función de costos el factor de vehículos para minimizar, mientras que Marco Polo define la optimización de este factor de forma implícita.

	#Viajes	#km. expresos	#Turnos	#Vehículos
PG42	66	220	12	5
IO	62	0	14	6
Marco Polo	66	40	11	5

Tabla 7.19: Escenario Con Reglas Laborales – Caso Héctor

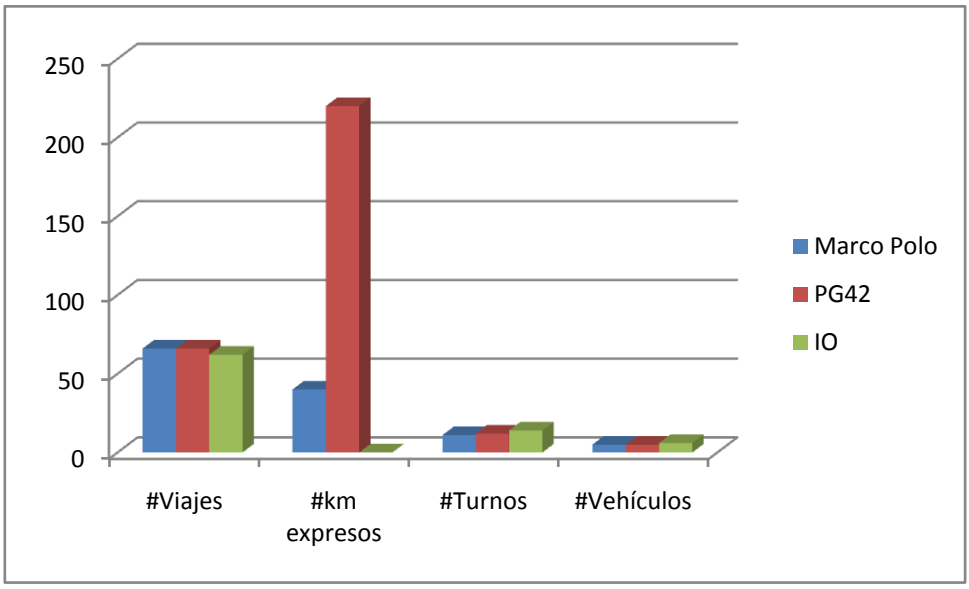

Figura 7.36: Gráfica comparativa - Escenario con reglas laborales

Como ya se mencionó anteriormente, en este escenario contamos con un resultado adicional para utilizar de referencia, el resultado del proyecto del grupo de IO. Cabe recordar que IO no consigue realizar un cubrimiento total de los viajes activos, en contraposición con PG42 y Marco Polo que sí lo consiguen.

La solución obtenida en Marco Polo con mejor aptitud, la cual será utilizada para realizar el análisis comparativo, contiene 40 kilómetros expresos y 11 turnos. En lo que refiere a los kilómetros expresos se observa, con respecto a la solución obtenida en PG42, una reducción del 82,82%, mientras que haciendo la comparación contra la solución obtenida en IO se observa un incremento en 40 kilómetros lo cual equivale a un 18,18% más si tomamos como referencia máxima al valor obtenido por el PG42.

Para el caso de los turnos se observa una mejoría en el valor de este factor en comparación con ambos resultados de referencia. En términos numéricos vemos una reducción en 1 turno con respecto al valor obtenido en PG42 y una reducción en 3 turnos respecto a la solución de IO. En cuanto a la cantidad de vehículos utilizados se mejora el resultado de IO al utilizar un vehículo menos que dicha solución y se mantiene el mismo valor alcanzado por el PG42.

Para concluir sobre la calidad final y general de las soluciones no podemos perder de vista que, pese a que Marco Polo incrementa levemente los kilómetros expresos con respecto a la solución obtenida por IO, este deja 4 viajes activos sin cubrir incurriendo en la violación a una restricción mucho más dura e importante que la de disminuir lo más posible los kilómetros expresos. Por lo tanto, en base a este análisis comparativo entre distintas soluciones para el mismo problema aplicado al caso de estudio Héctor, se concluye que la solución Marco Polo mejora los resultados obtenidos por los proyectos PG42 e IO.

7.4.2 Comparaciones – Caso COPSA

	#Viajes	#km. expresos	#Turnos	#Vehículos
PG42	1069	10010	300	115
Marco Polo	1069	5551	385	157

Tabla 7.20: Escenario Sin Reglas Laborales – Caso COPSA

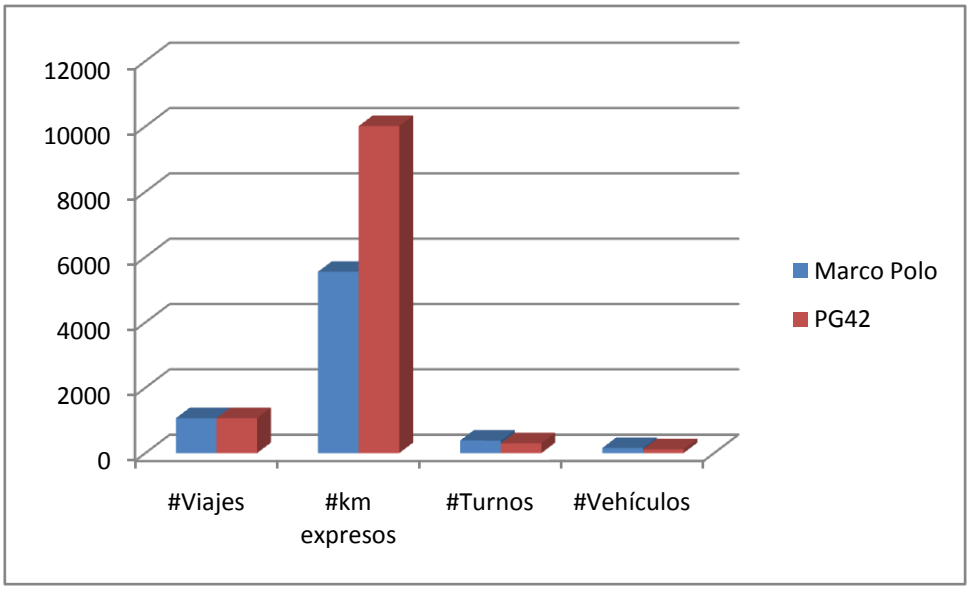

Figura 7.37: Gráfica comparativa - Escenario sin reglas laborales

Para el escenario sin reglas laborales se puede ver una reducción en el factor de kilómetros expresos con respecto al PG42 en un 44,55%. Sin embargo no se logran mejorar la cantidad de turnos, en la cual se presenta un incremento de un 28,33% con respecto al PG42, ni la cantidad de vehículos, en la cual el incremento es de un 36,52%.

	#Viajes	#km. expresos	#Turnos	#Vehículos	Tiempo Ejecución (hh:mm)
PG42	1069	29664	234	183	09:43
IO	997	4012	466	233	17:00
COPSA	1069	565	480	240	N/A
Marco Polo	1069	4820	453	150	10:32

Tabla 7.21: Escenario Con Reglas Laborales Caso COPSA

Finalmente se considera la solución de Marco Polo obtenida con mejor aptitud para realizar el análisis comparativo, esta vez, con tres valores de referencia. Dicha solución contiene 4820 kilómetros expresos y 453 turnos.

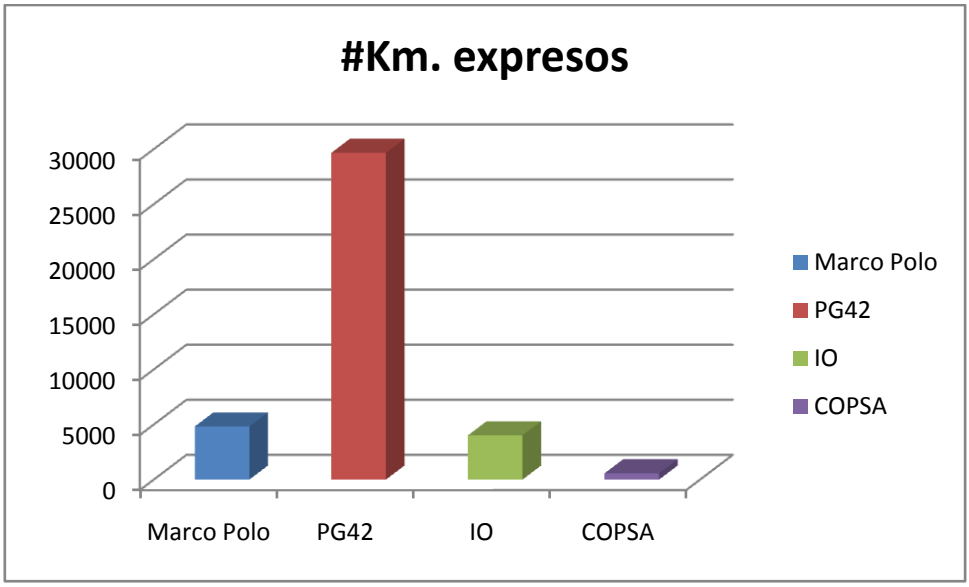

Figura 7.38: Gráfica comparativa de cantidad de km. Expresos - Escenario Con Reglas Laborales

En lo que refiere a los kilómetros expresos, tomaremos como valor base el resultado obtenido por COPSA el cual coincide con el menor valor encontrado para este factor. En relación al PG42, Marco Polo presenta la reducción, considerablemente alta, de un 85,38% en el valor obtenido para los kilómetros expresos, mientras que haciendo la comparación con los resultados obtenidos en IO y en COPSA se observan incrementos de 2,77% y 14,62% respectivamente.

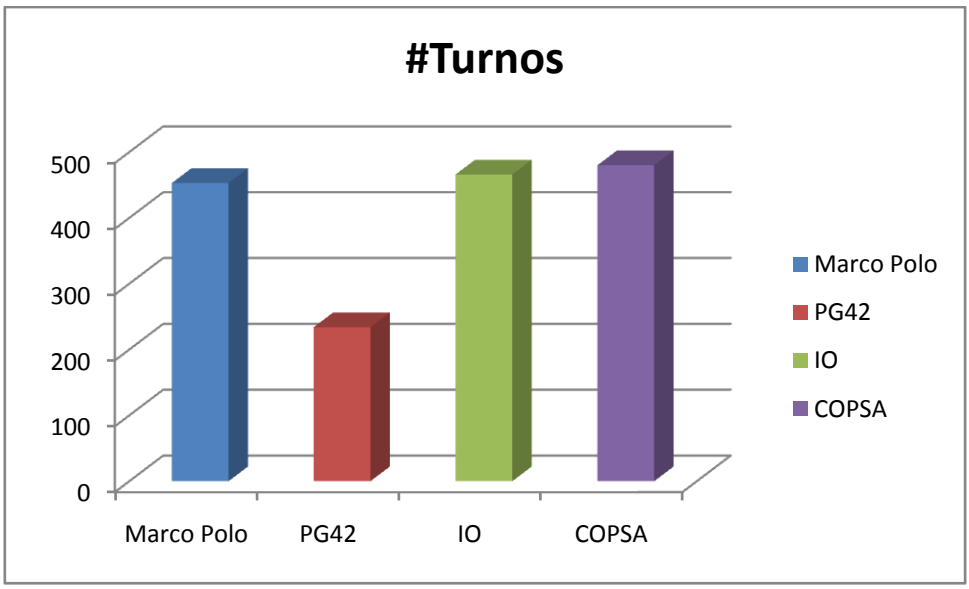

Figura 7.39: Gráfica comparativa de cantidad de turnos - Escenario Con Reglas Laborales

Analizando el factor turnos, y tomando nuevamente como base el valor obtenido por COPSA, vemos que Marco Polo logra reducir este valor en un 5,63%. Por su parte el Proyecto IO reduce el valor de COPSA en un 2,92% y el PG42 en un 51,25%. En este factor es en donde muestra mejor desempeño el proyecto PG42.

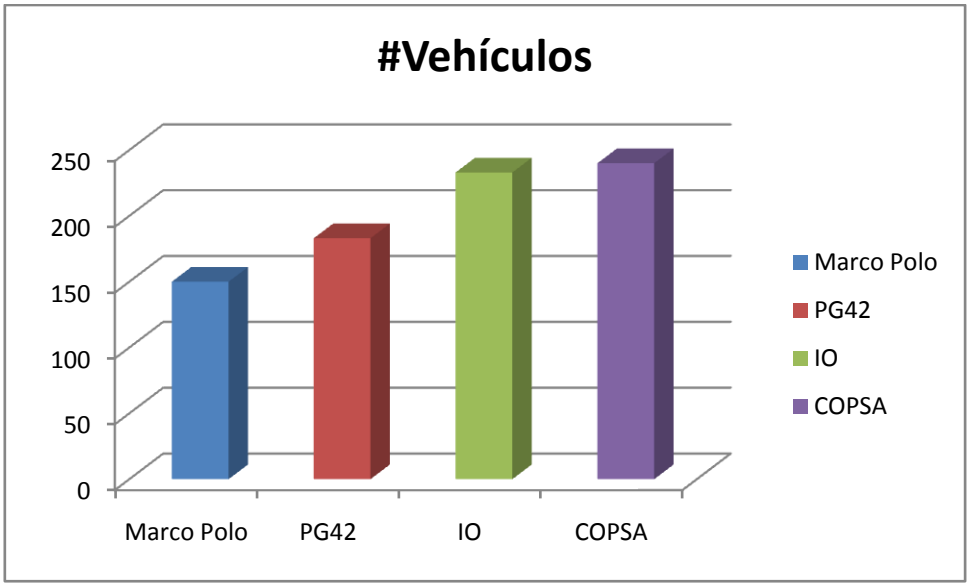

Figura 7.40: Gráfica comparativa de cantidad de vehículos - Escenario Con Reglas Laborales

Realizando un análisis similar para la cantidad de vehículos utilizada, esta vez es Marco Polo quién muestra el mejor desempeño reduciendo este valor con respecto a COPSA en un 37,5%, mientras que los proyectos IO y PG42 presentan mejorías de 2,92% y 24,75% respectivamente.

Adicionalmente, para el escenario con reglas laborales resolviendo el caso de estudio COPSA, contamos con los tiempos de ejecución empleados por cada uno de los sistemas. Se aprecia un tiempo de ejecución similar al del PG42 y notablemente menor al del Proyecto IO. No se cuenta con esta métrica para COPSA aunque se sabe que es del orden de semanas.

En base a este análisis comparativo entre distintas soluciones para el mismo problema, aplicado al presente caso de estudio, se concluye que la solución de Marco Polo logra mejorar la mayoría de los factores, considerados para determinar la calidad de la solución, con respecto a COPSA. El único factor que no se logra igualar o reducir es el de kilómetros expresos. En cuanto al PG42 se lograron mejorar todos los aspectos salvo el valor de los turnos. En este factor el PG42 muestra una reducción considerable en comparación con Marco Polo, con el alto costo de aumentar desmedidamente la cantidad de kilómetros expresos empleada. Si comparamos con la solución de IO lo primero que salta a la vista es que, para este caso de estudio, IO deja sin cubrir 72 viajes activos. Aún así Marco Polo logra mejorar casi todos los factores involucrados en la calidad de la solución, con excepción del factor de kilómetros expresos el cual presenta una diferencia mínima a favor de la solución de IO.

La mejor solución debe ser aquella que minimice los costos operativos de la empresa dado que esto implicará una maximización de las ganancias. Observando los costos salariales de los trabajadores así como el costo que insume tener operativo un vehículo, puede suceder que en un determinado momento resulte conveniente realizar más kilómetros expresos en beneficio de reducir turnos mientras que en otra realidad puede ser más beneficioso aumentar la cantidad de turnos con el fin de disminuir los kilómetros expresos. En base a este análisis queda claro que la mejor

solución no es aquella que tiene la menor cantidad de kilómetros expresos o la que tiene menor cantidad de turnos sino que la mejor solución queda dependiente del escenario en el cual está inmersa la empresa.

Capítulo 8: Conclusiones y trabajos futuros

8.1 Evaluación de resultados

Se logró diseñar e implementar una solución que resuelve el problema de generación de libros de servicios de transporte público de pasajeros de manera integrada en lo que refiere al ordenamiento de la flota y a la asignación de turnos. Dicho diseño modela el problema de forma tal que permite no solamente resolver el problema planteado en las condiciones establecidas sino que describe una manera diferente de pensar y observar el mismo a como se había enfocado en los proyectos antecesores. A su vez la solución persigue y respeta los fundamentos esenciales de los algoritmos genéticos en donde la búsqueda de la mejor solución se basa pura y exclusivamente en la aplicación de operadores genéticos. A su vez el diseño general del proyecto es modular y flexible permitiéndosele agregar nuevos componentes o mejorar los ya existentes. Entre los módulos diseñados e implementados se destacan el generador de soluciones iniciales, el cual cuenta con un lenguaje de instrucciones que permite especificar la manera en la cual serán construidas las soluciones iniciales del problema. A su vez otro que se destaca es el módulo de reglas laborales el cual permite que un usuario programador especifique dichas reglas.

En cuanto a la mejora de los resultados en comparación con las soluciones antes desarrolladas tanto a nivel de proyectos de grado como investigaciones del Departamento de Investigación Operativa, se concluye que los resultados obtenidos mejoran en general las soluciones encontradas. Para el caso de estudio Héctor definitivamente se mejoran los resultados anteriores y para el caso de estudio COPSA es discutible la decisión sobre cual solución computacional entre las presentadas en este proyecto es la mejor.

En definitiva, más que los resultados numéricos obtenidos en los casos de estudio, el principal resultado que arroja el presente proyecto es el modelado de la realidad planteada, la cual permite describir el problema de manera tal que hace posible la resolución integrada del problema.

En lo que refiere al enfoque adoptado para la resolución del problema de generación de libros de servicios, los algoritmos genéticos confirman ser una buena alternativa debido a la eficiencia que estos poseen para resolver problemas de tipo NP-difícil. Si bien es sabido que no garantizan la obtención de la solución óptima, por lo general obtienen soluciones aceptables en el contexto donde se aplican.

8.2 Posibles extensiones y trabajos futuros

Las extensiones que parecen más naturales de vislumbrar son las que surgen a partir de las posibilidades de expansión que ofrece cada uno de los módulos que componen el proyecto. Precisamente este era uno de los cometidos del diseño modular del sistema. En el caso del MRL, las extensiones pasan por definir nuevas reglas laborales o nuevas estrategias de pago (jornal, mensual, a destajo, etc.) lo cual lo puede hacer un usuario programador, con conocimientos del lenguaje C++, de manera sencilla y su procedimiento está detallado en el Anexo C. En lo que respecta al GSI, las extensiones

pueden incorporarse o bien mediante la implementación de nuevas instrucciones y/o condiciones o bien simplemente reutilizando las instrucciones ya implementadas y diagramando nuevos archivos de entrada que las invoquen y las hagan interactuar de la manera deseada, también por un usuario con conocimientos en programación.

La codificación de la solución diseñada en Marco Polo también ofrece posibilidades de expansión, ya que su estructura ofrece más posibilidades de uso de las que realmente se aprovechan en este proyecto. Un ejemplo de esto es que, por más de que en la práctica cada recorrido de turno tiene su recorrido de vehículo, que de alguna manera lo acompaña a lo largo de todo su trayecto, internamente los ruteos de vehículos están modelados por separado de los ruteos de turnos (Sección 5.1). Esto significa que, la codificación de la solución permite modelar que, un turno pueda hacer uso de más de un vehículo. Otra posibilidad que ofrece esta codificación es la de definir nuevos PET en lugares estratégicos para implementar, por ejemplo, puntos de relevo en donde se permita realizar un cambio de chofer, los cuales podrían ser definidos o bien de manera fija, indicando el lugar geográfico, o bien de manera relativa al comienzo o fin de un viaje. Los beneficios asociados a poder incorporar puntos de relevo tienen que ver con la incorporación de flexibilidad en la asignación de personal, lo cual puede redundar en una disminución de costos por ese concepto.

Por otro lado, el uso de la biblioteca Mallba como motor de esqueletos algorítmicos, ofrece la posibilidad agregarle paralelismo a la ejecución del algoritmo facilitándole al usuario la implementación paralela a partir de la implementación secuencial. Es posible configurar Mallba para que ejecute en paralelo como un sistema distribuido admitiendo además configurar conexiones tanto LAN como WAN. Esto podría generar una disminución en los tiempos de ejecución y quizás otorgaría una holgura a la hora de definir los parámetros, por ejemplo, permitiendo definir poblaciones más numerosas o ejecutar un número mayor de generaciones.

Anexo A. Manual de Instalación y Configuración

A.1. Introducción

El proyecto Marco Polo utiliza bibliotecas externas con lo cual es requisito previo que éstas sean instaladas y configuradas para poder desplegar y ejecutar la aplicación.

A.2. Objetivos

El objetivo de este anexo es esquematizar los pasos necesarios para instalar y configurar Marco Polo.

A.1. Instalación y configuración de MPICH2

MPICH2 (17) es una implementación del protocolo MPI (Message Passing Interface).

Para instalar MPICH2 se deben seguir los siguientes pasos:

- Obtener el paquete de la biblioteca desde el sitio oficial (17)
- Descomprimir el paquete en un directorio
- Crear el directorio en donde será instalado MPICH2

```
mkdir /ruta_mpich2/mpich2-install
```

- En una terminal posicionarse en el directorio en donde fue descomprimido el paquete
- Ejecutar la siguiente secuencia de comandos:

```
./configure -prefix=/ruta_mpich2/mpich2-install --disable-f77 --
disable-fc
make |& tee make.log
make install |& tee install.log
```

A.2. Instalación y configuración de MaLLBa

Requisitos para la instalación de MaLLBa

- Tener instalado MPICH2

Para instalar la biblioteca MaLLBa se deben seguir los siguientes pasos:

- Obtener la biblioteca desde el sitio oficial del proyecto (16)
- Descomprimir en un directorio el paquete mallba.tar.gz
- Configurar la instalación de MaLLBa editando el archivo environment. En particular se debe especificar:
    - la ruta en donde será instalada la biblioteca MaLLBa (variable MALLBA_DIR)
    - la ruta en donde se encuentra instalado MPICH2 (variable MPI_BIN)
- En una terminal posicionarse en el directorio donde fue descomprimido el paquete

- Ejecutar el siguiente comando:

```
make all
```

A.3. Instalación y configuración de Mini-XML

Mini-XML (18) es una biblioteca que brinda funcionalidades para trabajar con archivos XML. Para instalar esta biblioteca se deben seguir los siguientes pasos:

- Obtener esta biblioteca desde el sitio oficial de Mini-XML (18) y descargar la versión 2.6
- Descomprimir el paquete en un directorio
- Crear el directorio en donde será instalado Mini-XML:

```
mkdir /ruta_minixml/minixml
```

- En una terminal posicionarse en el directorio en donde fue descomprimido el paquete
- Ejecutar la siguiente secuencia de comandos:

```
./configure --prefix=/ruta_minixml/minixml
make
make install
```

A.4. Despliegue de la aplicación

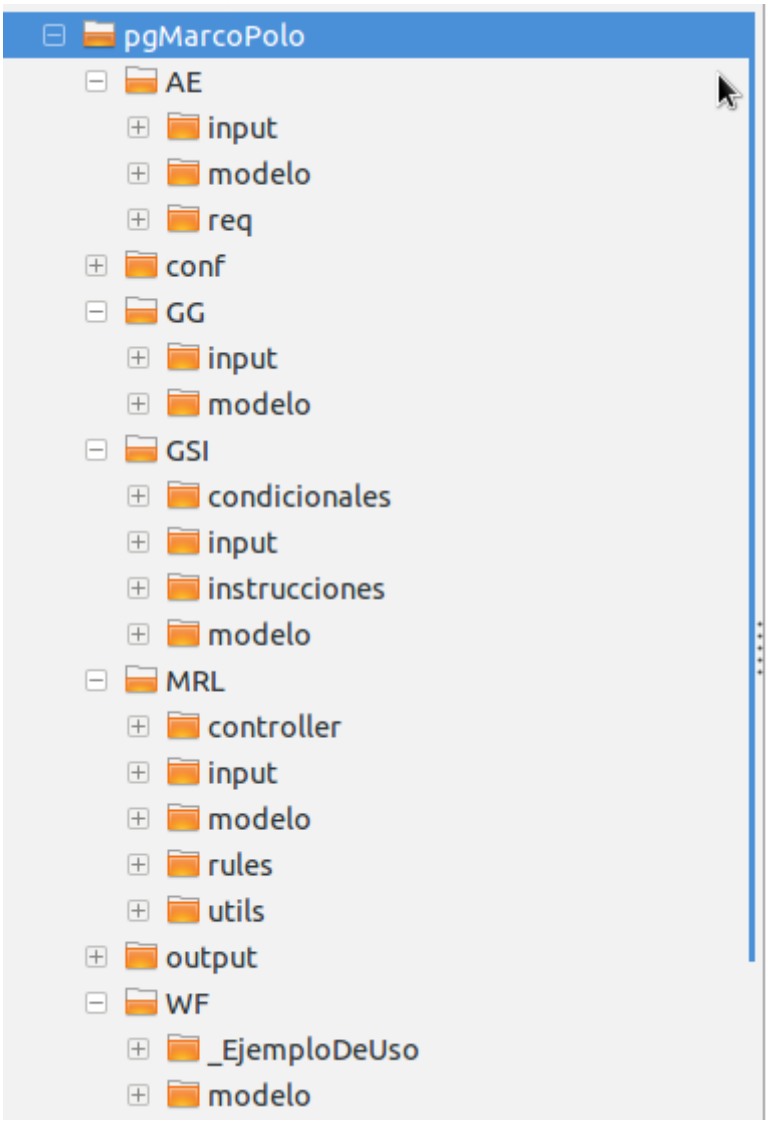

Figura A.1: Estructura del árbol de directorios

- AE/input/: en este directorio se encuentra el archivo MarcoPoloGA.cfg el cual contiene la configuración de los parámetros necesarios para la ejecución del algoritmo genético sobre la biblioteca MaLLBa. Entre los parámetros que se deben definir se destacan:
  - cantidad de ejecuciones independientes
  - cantidad máxima de generaciones
  - cantidad de individuos por generación
  - cantidad de descendientes por generación
  - algoritmo de selección de progenitores
  - algoritmo de selección de descendientes
  - probabilidades de los operadores de mutación y cruzamiento

- conf/: este directorio contiene la clase ConfiguracionGlobal y el archivo Config.cfg el cual contiene los siguientes parámetros:

  - configAE: ruta del archivo de configuración del módulo AE (requerido por Mallba para el funcionamiento del algoritmo genético)
  - configGG: ruta del archivo que contiene los datos del problema a resolver
  - semilla: semilla a utilizar en el generador de números pseudo-aleatorios
  - configGSI: ruta del archivo de configuración del modulo GSI
  - reglasLaborales: este parámetro toma los valores SI ó NO. Para el caso del primer valor las reglas laborales son tenidas en cuenta por la aplicación, en el otro caso no
  - imprimirSoloBestCost: este parámetro toma los valores SI ó NO
  - imprimirMejorSolucion: este parámetro toma los valores SI ó NO. Si al parámetro imprimirSoloBestCost se le estableció el valor SI, entonces se ignora este parámetro
  - imprimirErroresGSI: Este parámetro toma los valores SI ó NO

- GG/input: directorio sugerido para almacenar los archivos que contengan la información del problema a resolver

- GSI/input: directorio sugerido para almacenar los archivos que contengan las instrucciones para generar las soluciones iniciales

- MRL/input: en este directorio se encuentra el archivo rules.xml. En dicho archivo se definen las reglas laborales que serán tenidas en cuenta al momento de buscar una solución al problema dado

- output/: archivo sugerido para re-direccionar la salida a archivo del resultado de la ejecución de la aplicación. Para re-direccionar la salida a archivo utilizar el siguiente comando:

```
./pgMarcoPolo  AE/input/input.cfg >> ./output/salida.out
```

A.5. Modo de ejecución de la aplicación

Para ejecutar la aplicación se debe ejecutar el comando:

```
pgMarcoPolo nom_archivo
```

donde el parámetro “nom_archivo” indica el archivo .cfg a usar, el cual contiene la información que configura el algoritmo genético. En caso de que no se pase ningún parámetro, por defecto la aplicación toma el archivo MarcoPoloGA.cfg del directorio relativo a la raíz de instalación AE/input/

Anexo B. Gran Grafo

B.1. Introducción

Si uno hace el ejercicio de abstraerse del problema de planificación operativa y se posiciona como un observador que ve el problema desde arriba, representado en un mapa, podría verse una malla compuesta por puntos que indican lugares geográficos determinados, los viajes activos y los posibles viajes expresos y espera que pueden realizarse a partir de todos los lugares geográficos involucrados, sobre la cual se terminará dibujando la solución final. El Gran Grafo (en adelante GG) representa dicha malla modelándola como un grafo en donde las parejas lugar-horario son los vértices y los distintos tipos de viajes son las aristas que navegan desde su vértice origen a su vértice destino. Este grafo será el soporte del algoritmo genético el cual le solicitará toda la información que necesite para armar los servicios.

B.2. Objetivos

Los objetivos principales del GG son, por un lado, ser el contenedor de toda la información referente a la realidad del problema y, por otro, oficiar de nexo entre el algoritmo y los diferentes módulos que componen el proyecto para poder resolver cuestiones relativas a la operativa del algoritmo.

B.3. Funcionamiento

El Gran Grafo será la primera estructura a crear, en la cual quedará almacenada toda la información referente al problema. Luego, el algoritmo y los distintos módulos, acudirán a él para hacerse con esta información.

Se comienza procesando el archivo de entrada con todos los viajes activos del problema deduciendo, para cada uno de ellos, los PET que tiene asociados. Cada PET se instanciará y almacenará en caso que no haya sido creado antes. Si, por el contrario, el PET ya existía se obtiene la instancia previamente creada para asociar luego, el nuevo viaje que parte o llega a dicho PET. A continuación se crea el Viaje y se asocia a sus dos PET correspondientes. Al finalizar este procesamiento se habrán creado todos los PET necesarios y todos los Viajes Activos.

Luego se crean los Viajes Espera agrupando en listas a los PET que pertenecen a la misma ciudad. Estas listas serán ordenadas de manera ascendente según el horario del PET. A continuación se itera sobre estas listas y se crea el viaje espera que une el PET de la posición i de la lista con el PET de la posición i+1.

Finalmente quedan por calcular los Viajes Expreso. Para esto se realizan dos iteraciones anidadas que iteran sobre i y j respectivamente. En cada una de estas iteraciones se revisa si desde el PET i se puede realizar un viaje expreso al PET j. Esto se deduce en base a la hora del PET i (primer término), el tiempo que llevaría hacer un viaje desde la ciudad a la que pertenece el PET i al PET j (segundo término) y la hora del PET j (tercer término). Si la suma del primer término con el segundo es menor al

tercero se puede afirmar que se trata de un viaje realizable entonces este viaje es creado como viaje expreso y almacenado en el Gran Grafo.

B.4. Diseño

El diseño de este módulo se basa principalmente en tres grandes clases que modelan los conceptos de viaje, PET y de Gran Grafo.

- *GranGrafo*: la clase *GranGrafo* oficia de controlador y contiene la colección de todos los viajes y PET creados así como de todas las ciudades involucradas en la planificación. En este controlador se almacenan además los datos necesarios para el cálculo de la función de costos como ser constantes con valores de precios o rendimientos, los cuales son levantados al comienzo a partir de un archivo de entrada.

- *Viaje*: la clase *Viaje* mantiene referencias a sus PET origen y destino y cuenta con toda la información inherente al propio viaje.

- *PET*: esta clase mantiene colecciones con los viajes que salen y parten de él diferenciadas por tipo de viajes, además de punteros al posible viaje espera que pueda salir de él como así también al que pueda llegar.

- *Ciudad*: se cuenta también con esta clase que modela el concepto de lugar geográfico y mantiene una colección con los PET que pertenecen a dicho lugar. Esta relación es bidireccional por lo que cada PET también tiene acceso a su ciudad.

B.5. Diagrama de clases de diseño

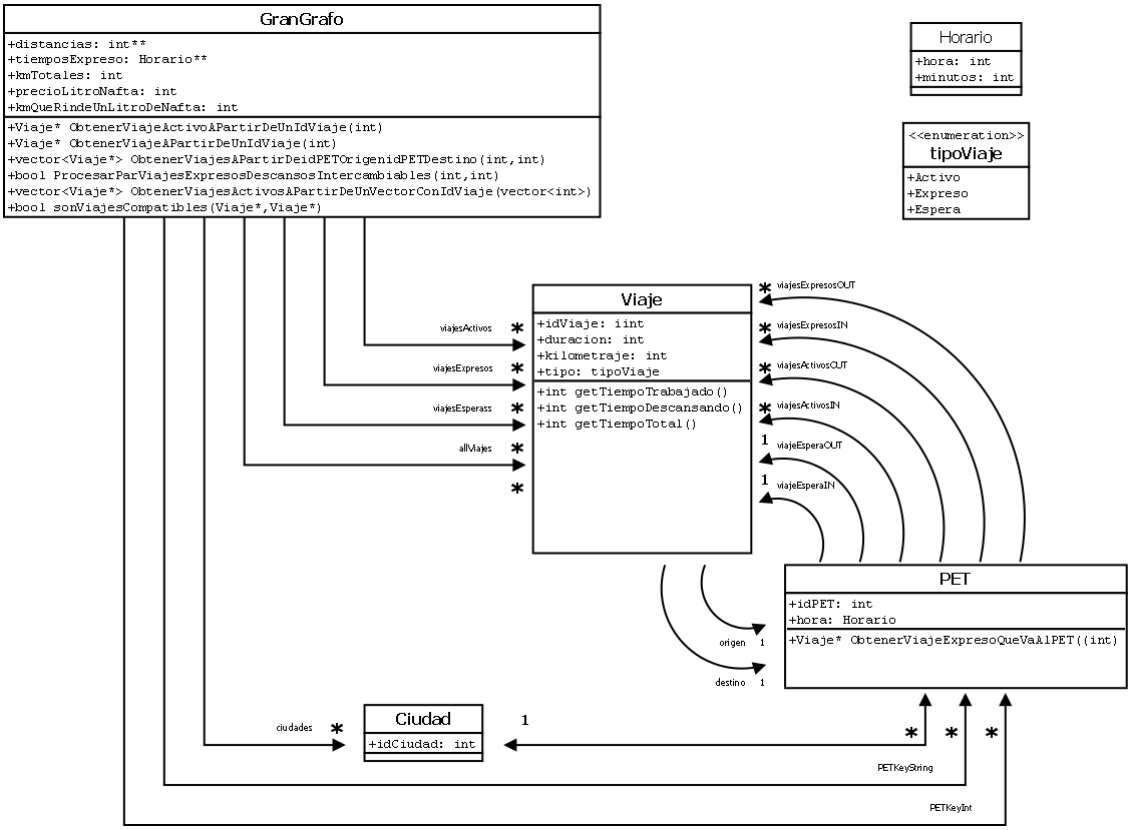

Figura B.8.1: Diagrama de clases de diseño

B.6. Datos de entrada

El GG se crea a partir de los datos contenidos en el archivo de entrada con toda la información relativa a la etapa previa de planificación en donde se definieron los viajes que deben ser realizados. Este archivo contiene los siguientes datos:

- *Cantidad de ciudades:* indica la cantidad total de ciudades que participan, ya sea como origen o destino de cualquier viaje activo
- *Matriz de distancias entre ciudades:* esta matriz debe ser el producto cartesiano entre todas las ciudades participantes de la instancia del problema indicando la distancia entre cada una de ellas medida en km.
- *Matriz de tiempos para viajes expresos:* esta matriz debe ser el producto cartesiano entre todas las ciudades participantes de la instancia del problema indicando la distancia entre cada una de ellas medida tiempo, en el formato hh:mm
- *Cantidad de viajes:* indica la cantidad de viajes activos del problema
- *Viajes:* se debe indicar ciudad origen, ciudad destino, hora de salida y hora de llegada (ambas en formato hh:mm)
- *Precio litro nafta:* indica el precio actual, en pesos, del litro de nafta
- *Km que rinde un litro de nafta:* indica la cantidad de kilómetros que rinde un litro de nafta en los vehículos utilizados en el problema

A continuación se detalla cómo debería ser el archivo de entrada para el GG en el ejemplo de la figura Figura 3.4 de la Sección 3.1.6

```
# Cantidad de ciudades
2
# Matriz de distancias entre las ciudades
0 32
32 0
# Matriz de tiempos para viajes expresos
0:00 0:50
0:50 0:00
# Cantidad de viajes
4
# Viajes, ciudad origen, destino, horaSal, horaLleg
0 1 8:00 9:00
1 0 8:00 9:00
0 1 9:00 10:00
1 0 10:00 11:00
# Precio litro nafta
35
# Km que rinde un litro de nafta
5
```

Anexo C. Módulo de Reglas Laborales

C.1. Introducción

Las reglas labores en un determinado país, región o estado pueden ser tan variadas y complejas como lo determinen las normativas vigentes. En particular, la realidad a representar en el presente proyecto quedará limitada a los documentos proporcionados por la empresa de transporte de pasajeros COPSA (Compañía de Ómnibus de Pando S.A.).

C.2. Objetivos

El módulo de reglas laborales, de aquí en mas denominado MRL, tiene como principal objetivo representar la realidad determinada por la normativa vigente en lo referente a las condiciones laborales que deben satisfacer tanto empleados como empleadores del transporte público de pasajeros. Además se desea que el MRL cumpla con las siguientes propiedades:

- Extensible: brindar la posibilidad de agregar nuevas reglas laborales y tipos de relaciones contractuales empresa-empleado
- Modular: permitir la reutilización del módulo en otros proyectos siendo independiente de las características del mismo
- Eficiente: realizar los cálculos que correspondan en el menor tiempo posible.

C.3. Diseño

Con el fin de lograr un módulo de reglas laborales que pueda ser reutilizable en futuros proyectos vinculados al transporte público de pasajeros se definen dos clases que implementan conceptos generales que aplican a cualquier realidad vinculada al problema:

- Actividad: una actividad hace referencia a una acción particular de un empleado durante la jornada. En dicha actividad se registra la cantidad de tiempo que el empleado estuvo trabajando y cuanto estuvo descansando.
- Turno: un turno se define como un grupo o conjunto de actividades. Semánticamente un turno significa la jornada laboral de un empleado.

En base a estos dos sencillos conceptos se construye el MRL.

C.3.1. Controlador

El componente que actúa de nexo o interface entre el MRL y el algoritmo que hará uso del mismo es el *WorkerRuleController*. Este controlador fue implementado siguiendo el patrón de diseño Singleton. Se destacan dos operaciones o métodos que el controlador provee:

- *isFeasible* el cual permite determinar dado un conjunto de turnos, si los mismos satisfacen las reglas laborales definidas o no

- *costoTurnos* computa dado un conjunto de turnos, el costo de los mismos en base a una estrategia de pago a los empleados previamente definida

C.3.2. Interfaces

El módulo cuenta con dos interfaces las cuales lo hacen extensible y adaptable a distintas realidades laborables.

- *ReglaLaboral:* esta interface modela el concepto de una norma laboral, con lo cual para que un turno sea válido debe satisfacer todas las reglas laborales que se definan.

- *EstrategiaPago:* esta interface modela el universo de relaciones empleador-empleado que puedan existir.

C.3.2.1. BreakRules

En base a la realidad bajo la cual se está rigiendo el presente proyecto se define el concepto *BreakRules* siendo este una implementación de la interface *ReglaLaboral*. Una *BreakRules* incluye al conjunto de *BreakRule* (concepto definido en el PG42) establecidas en el problema. Con la regla laboral *BreakRules* se busca comprobar que, dado un determinado turno, este satisface al menos uno de los descansos posibles definidos (*BreakRule*).

Tras diferentes pruebas realizadas decidimos definir ciertas variables que consideramos necesarias para suavizar restricciones de tiempo referentes a los turnos.

Se observó que la probabilidad de obtener turnos que satisfagan la regla laboral *BreakRules* era muy baja. Por dicho motivo se definieron las siguientes variables de holgura para aumentar la probabilidad de obtener turnos factibles:

- *MinutosMinimoDescanso:* esta variable hace referencia a la duración mínima en minutos que puede tener un descanso. Se define para que pequeños períodos de tiempo en los cuales no se está trabajando no sean considerados como descansos

- *CotaInferiorToleranciaDescanso y CotaSuperiorMinutosDescanso*: estas variables se utilizan junto a la propiedad *duration* del concepto *BreakRule* para construir un intervalo de tiempo que determina la duración de un descanso válido. Se define relajar la duración fija del descanso permitiendo que el mismo pueda tener una cantidad de minutos de menos o de más a modo de tolerancia.

Se observó también que para aquellos turnos que no llevan una carga horaria total (en adelante *mini-turnos*), la factibilidad de los mismos no se consigue al evaluar BreakRules. Por dicho motivo se añadieron las siguientes variables:

- *MaxMinutosMiniTurno:* esta variable indica la cantidad máxima en minutos que puede tener un mini-turno

- *MinPorcentajeDescansoMiniTurno:* esta variable hace referencia al mínimo porcentaje de tiempo, en relación al total, que debe ser tiempo de descanso

- *MaxPorcentajeDescansoMiniTurno:* análogamente a la variable anterior, esta hace referencia al máximo porcentaje de tiempo, en relación al total, que puede ser tiempo de descanso

- *MinutosMinimoDescansoMiniTurno:* esta variable es análoga a *MinutosMinimoDescanso* y se utiliza con el mismo fin

C.3.2.2. Jornal

El otro concepto definido atendiendo a la realidad actual fue el de Jornal. Dicho concepto implementa la interface *EstrategiaPago*. En el contexto del presente proyecto los empelados de la compañía de transporte público son jornaleros, es decir que cobran por día trabajado. En base a esto, para calcular el costo de un determinado turno se definen las siguientes variables:

- *DuracionJornal:* esta variable hace referencia a la duración en minutos de un jornal

- *ValorJornal:* análogamente a la anterior esta variable refiere al valor monetario de un jornal

- *MultiplicadorMinutosExtras*: esta variable es un coeficiente que indica por cuanto se deben multiplicar los minutos extras trabajados. A modo de ejemplo, suponiendo que se hicieron 60 minutos extras y el coeficiente es 2, entonces al momento de calcular el costo del turno se considerará que la duración del mismo responde al siguiente cálculo: duracion_joranal + 2*60.

En caso de que un turno tenga una duración menor a la duración definida para el jornal, al momento de calcular el costo del turno, se le asignará la duración del jornal definida como medida de penalización por ser un turno con duración menor al esperado.

C.3.3. Diagrama de clases de diseño

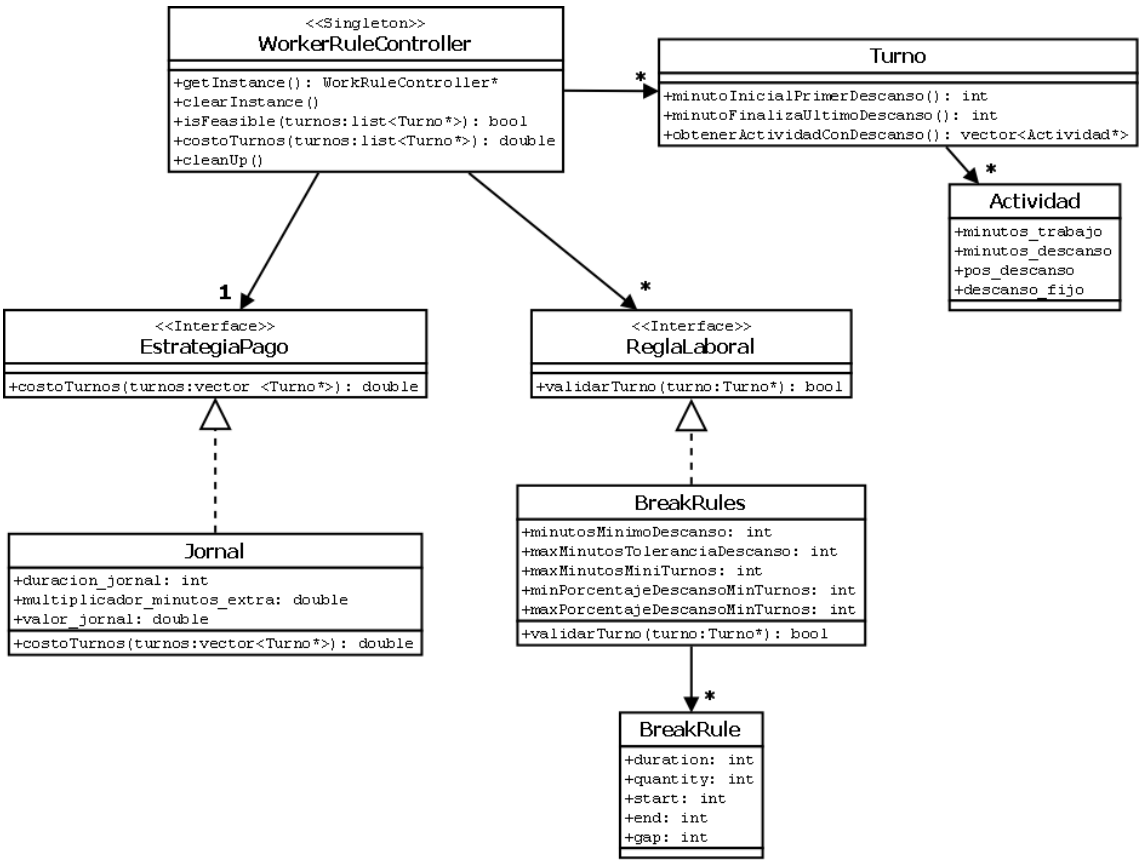

Figura C.8.2: Diagrama de clases de diseño

C.3.4. Despliegue del Modulo de Reglas Laborales

El módulo de reglas laborales se despliega de acuerdo a la jerarquía que se muestra en la siguiente figura.

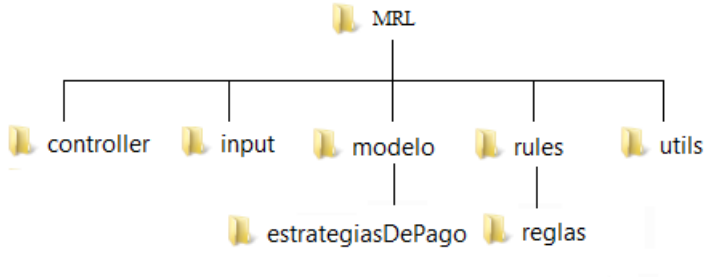

Figura C.8.3: Estructura de directorios del MRL

- *controller*: contiene el código fuente del *WorkerRuleController*
- *input*: en él se encuentra el archivo rules.xml
- *modelo*: allí se encuentran los archivos fuentes de los conceptos de actividad, estrategia de pago y turno

- *estrategiasDePago*: lugar para almacenar los archivos fuentes de las estrategias de pago definidas
- *rules*: contiene la interface *ReglaLaboral*
- *reglas*. bajo este directorio deben guardarse los archivos fuentes de las reglas laborales definidas
- *utils*: directorio que contiene código que implementa funcionalidades internas para el correcto funcionamiento del modulo.

C.3.5. Definición de reglas laborales.

La reglas laborales se deben definir en el archivo rules.xml ubicado en la carpeta input dentro la jerarquía de despliegue del módulo. Existen dos grandes conceptos a modelar, las estrategias de pago, en las cuales se especifica todo lo referente al pago de los jornales y las reglas laborales propiamente dichas. Para definir estos conceptos se deben seguir las instrucciones que se detallan en la presente sección.

C.3.5.1. Definición de estrategias de pago

Dentro de los tags <EstrategiaDePago> se definen cada una de las estrategias de pago que se quieran utilizar. A continuación se muestra un ejemplo:

```
<EstrategiaDePago>
	<EstrategiaDePago_1>
		<Propiedad_1_1> dato_1_1 </Propiedad_1_1>
		<Propiedad_1_2> dato_1_2 </Propiedad_1_2>
	</EstrategiaDePago_1>
	<EstrategiaDePago_2>
		<Propiedad_2_1> dato_2_1 </Propiedad_2_1>
	</EstrategiaDePago_2>
</EstrategiaDePago>
```

En la función *init* de la clase *WorkerRuleController* se debe instanciar de manera explícita la estrategia de pago a utilizar. Por ejemplo, si se quiere utilizar la estrategia de pago *EstartegiaDePago_1* se debe agregar la siguiente línea de código a dicha función:

```
this->estrategia_de_pago = new EstartegiaDePago_1 ();
```

A modo de aclaración se deja constancia que solamente puede existir una única estrategia de pago instanciada.

Es responsabilidad del desarrollador implementar en el constructor de toda clase descendiente de *EstrategiaDePago* la lectura de las propiedades almacenadas en el archivo *rules.xml* relacionadas con su estrategia de pago.

C.3.5.2. Definición de reglas laborales

Las reglas laborales pueden ser agregadas en cualquier parte del documento rules.xml siempre que sean declaradas como descendientes directas del tag principal <*Rules*>. A continuación se muestra un ejemplo:

```
<Rules>
```

```
      <ReglaLaboral_1>
      <Propiedad_1_1>
            <Propiedad_1_1_1> dato_1_1_1 </Propiedad_1_1_1>
            <Propiedad_1_1_2> dato_1_1_2 </Propiedad_1_1_2>
      </Propiedad_1_1>
      </ReglaLaboral_1>
      <ReglaLaboral_2>
            <Propiedad_2_1> dato_2_1 </Propiedad_2_1>
      </ReglaLaboral_2>
      <ReglaLaboral_3> dato_3 </ReglaLaboral_3>
<Rules>
```

En la función *init* de la clase *WorkerRuleController* se deben instanciar de manera explícita las reglas laborales que se quiere cargar al sistema. Por ejemplo si se quieren utilizar las reglas laborales *ReglaLaboral_1, ReglaLaboral_2* y *ReglaLaboral_3* se deben agregar las siguientes líneas de código:

```
ReglaLaboral* rl1 = new ReglaLaboral_1 ();
ReglaLaboral* rl2 = new ReglaLaboral_2 ();
ReglaLaboral* rl3 = new ReglaLaboral_3 ();
this->reglas_laborales.push_back(rl1);
this->reglas_laborales.push_back(rl2);
this->reglas_laborales.push_back(rl3);
```

C.3.5.3. Reglas en Marco Polo

A continuación se presenta el archivo rules.xml utilizado en el presente proyecto.

```
<Rules>
      <EstrategiaDePago>
            <Jornal>
                  <DuracionJornal>480</DuracionJornal>
                  <MultiplicadorMinutosExtras>2</MultiplicadorMinutosExtras>
                  <ValorJornal>1000</ValorJornal>
            </Jornal>
      </EstrategiaDePago>
      <BreakRules>
            <MaxMinutosMiniTurno>240</MaxMinutosMiniTurno>
      <MinutosMinimoDescansoMiniTurno>6</MinutosMinimoDescansoMiniTurno>
      <MinPorcentajeDescansoMiniTurno>0</MinPorcentajeDescansoMiniTurno>
      <MaxPorcentajeDescansoMiniTurno>50</MaxPorcentajeDescansoMiniTurno>
            <BreakRule>
                  <duration>15</duration>
                  <quantity>3</quantity>
                  <start>10</start>
                  <end>480</end>
                  <gap>20</gap>
                  <MinutosMinimoDescanso>11</MinutosMinimoDescanso>
            <CotaSuperiorMinutosDescanso>11</CotaSuperiorMinutosDescanso>
      <CotaInferiorToleranciaDescanso>7</CotaInferiorToleranciaDescanso>
            </BreakRule>
            <BreakRule>
                  <duration>50</duration>
                  <quantity>1</quantity>
                  <start>10</start>
                  <end>480</end>
                  <gap>1</gap>
                  <MinutosMinimoDescanso>11</MinutosMinimoDescanso>
```

```
            <CotaSuperiorMinutosDescanso>26</CotaSuperiorMinutosDescanso>
      <CotaInferiorToleranciaDescanso>20</CotaInferiorToleranciaDescanso>
            </BreakRule>
            <BreakRule>
                  <duration>20</duration>
                  <quantity>2</quantity>
                  <start>10</start>
                  <end>480</end>
                  <gap>1</gap>
                  <MinutosMinimoDescanso>11</MinutosMinimoDescanso>
            <CotaSuperiorMinutosDescanso>18</CotaSuperiorMinutosDescanso>
      <CotaInferiorToleranciaDescanso>6</CotaInferiorToleranciaDescanso>
            </BreakRule>
      </BreakRules>
</Rules>
```

Anexo D. Generador de Soluciones Iniciales

D.1. Introducción

El GSI es un módulo específico para crear soluciones iniciales factibles. Dentro de este módulo se encapsula toda la lógica necesaria para tal fin. El GSI es asistido por otros módulos para proveer al algoritmo genético de individuos factibles, una funcionalidad clave y generalmente costosa en este tipo de metaheurísticas.

D.2. Objetivos

El principal objetivo de GSI es proveer la población inicial de individuos factibles al algoritmo genético para comenzar su ejecución.

D.3. Modelo

La función principal del GSI es armar un libro de servicios. El libro de servicios está compuesto por muchos servicios y estos por muchos recorridos. La clase recorrido servirá tanto para modelar los recorridos de los vehículos como los recorridos de los turnos. La clase recorrido tendrá visibilidad sobre la clase viaje definida en el GG lo cual, a su vez, implica una visibilidad sobre las clases PET y Ciudad. Todos los viajes activos estarán asociados a un único recorrido, mientras que los viajes expresos y espera estarán asociados a uno, muchos, o a ningún recorrido.

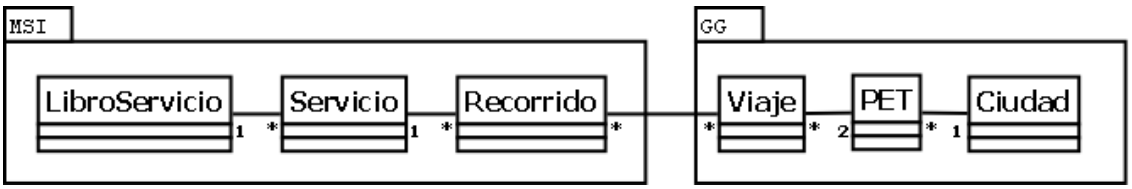

Figura D.1: Diagrama conceptual

Por otro lado, un recorrido pertenece a un único servicio, pero un servicio puede tener varios recorridos. Cada uno de estos recorridos, por si solos, serán la representación de los viajes que realiza un turno. Luego, el recorrido resultado de la concatenación de los turnos de un servicio con el eventual agregado de los viajes espera necesarios para que el recorrido quede factible, será el recorrido del vehículo asociado a dichos turnos. En esta etapa no nos interesa saber cuál es el vehículo específico que se asocia al servicio, ni tampoco saber a qué chofer en concreto se asocia el recorrido. Cabe destacar que los viajes que se puedan necesitar agregar para formar el recorrido del vehículo deben ser viajes espera dado que, en dichos huecos horarios, no hay chofer que traslade el vehículo. Esto trae aparejado que, en un servicio, los recorridos de turnos consecutivos a concatenar deben terminar y comenzar en una misma ciudad para poder ser unidos. Esta característica, junto con la condición de que cada recorrido de turno sea factible, es lo que determinará si un servicio es factible o no.

Finalmente tendremos que un servicio pertenece a un único libro de servicios. La cantidad de instancias de servicios que existan indicará la cantidad de vehículos que se

utilicen en dicho libro de servicios, mientras que la cantidad de instancias que existan de recorridos de turnos indicará la cantidad de turnos.

El controlador de este módulo se llama *GeneradorSolucionInicial* y su principal y única función es *generarSolucionInical* que devuelve un libro de servicios en caso de que sea factible de construir y en caso contrario retornará null. Esta función es dinámica, en el sentido de que según como sea la configuración que se haya prefijado será la forma en que se procederá para construir el libro de servicios. Cada invocación a esta función obtendrá un libro de servicios diferente ya que en su implementación se utiliza aleatoriedad a la hora de tomar ciertas decisiones.

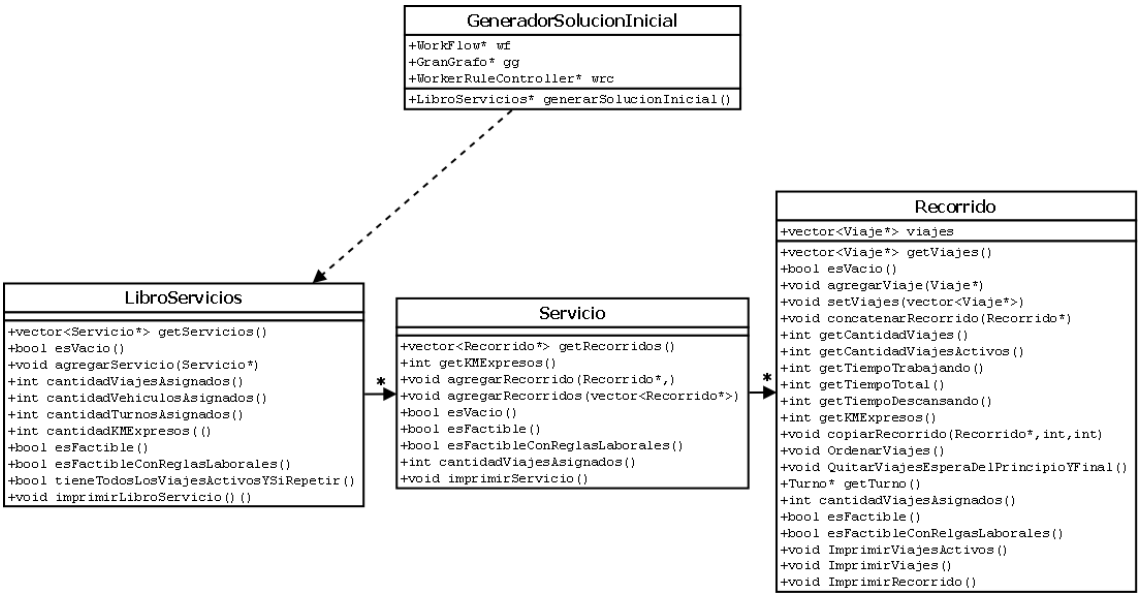

Figura D.2: Diagrama de clases

D.4. Interacción

Para cumplir con sus objetivos el GSI consume funcionalidades provenientes de los módulos: Gran Grafo, Módulo de Reglas Laborales y Work Flow.

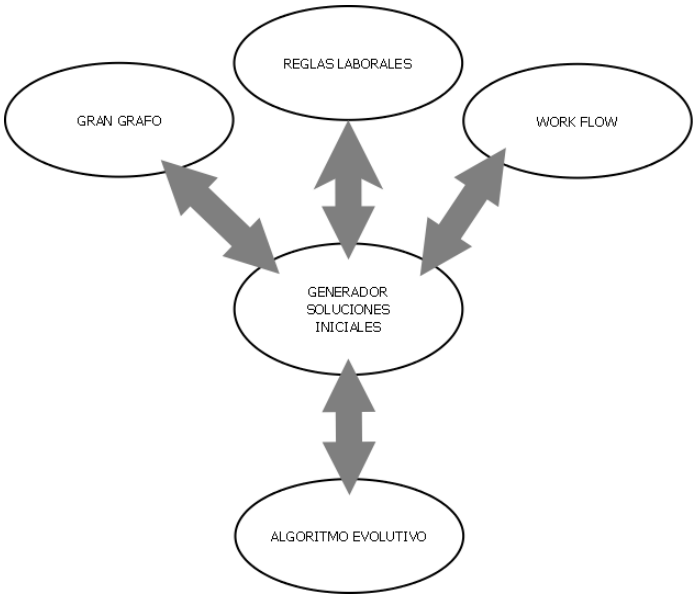

Figura D.3: Interacciones

- GG: esta interacción está motivada por el hecho de que la información sobre los viajes que debe contener cada solución inicial se encuentra contenida en el GG. El GSI necesita saber cuáles son los viajes activos que deben ser cubiertos junto con sus horarios y lugares de partida y llegada, necesita hacer uso de la matriz de viajes compatibles y obtener datos como los tiempos entre diferentes ciudades. El GSI consumirá estas funcionalidades del GG y además importará sus estructuras de datos: Viaje, PET y Ciudad.

- MRL: este módulo se hace necesario a la hora de saber si los turnos creados son factibles o no en cuanto a reglas laborales y de descanso. Para poder comunicarse con dicho módulo es necesario cargar cada turno generado en una estructura Turno

- WF: este módulo se utiliza para orquestar la secuencia de acciones a ejecutar para generar la población inicial. De hecho la motivación para crear el módulo WF fue el surgimiento de una necesidad de flexibilidad en el GSI. Este módulo se presenta al detalle en el anexo E.

D.5. Algoritmo

La técnica que se utiliza en el GSI para armar el libro de servicios se aproxima mucho a un algoritmo greedy. Para construir una solución el algoritmo va realizando iteraciones en donde en cada una de ellas se construye parte de la solución hasta completarla. La diferencia es que la técnica greedy busca en cada paso una solución local óptima mientras que en el GSI se utiliza cierta aleatoriedad para ir armando la solución.

En cada iteración se tiene como entrada un conjunto de viajes activos a cubrir y como salida un conjunto de servicios. En la primera iteración la entrada está dada por todos los viajes activos, en la segunda iteración la entrada serán los viajes activos que aún no fueron asignados y así sucesivamente. A medida que avanzan las iteraciones, cada salida irá incrementando la cantidad de servicios generados y cuando se termine la

ejecución tendremos una solución factible que podrá ser utilizada por el algoritmo genético como un individuo de la población inicial.

La implementación del algoritmo está basada en la ejecución de distintas instrucciones y condiciones las cuales están modeladas como entidades independientes que realizan acciones concretas. Luego, con un archivo de texto plano como entrada y la asistencia del módulo de WF, se definen las variables a usar y se especifica la secuencia concreta de instrucciones y condiciones a ejecutar junto con los eventuales saltos e iteraciones que deban existir en dicha secuencia. Este archivo deberá respetar un formato específico en base a la sintaxis definida en el Anexo E para poder hacer uso del módulo de WF.

A continuación se presentarán los detalles de implementación de las instrucciones y condiciones implementadas en Marco Polo para obtener soluciones iniciales para el problema planteado.

D.5.1. Instrucciones

- *ObtenerUnRecorrido_MetodoSinExpresos*: Esta función tiene como objetivo armar un recorrido que contenga cantidad de Km. expresos nula. Recibe como entrada la lista de viajes activos que están disponibles, esto es, que todavía no fueron cubiertos por la solución a la cual pertenecerá este recorrido. Se comienza sorteando una ciudad y obteniendo el PET asociado a esta ciudad con el menor horario. A continuación se selecciona un viaje cualquiera de entre el conjunto de viajes activos que parten desde este PET. El viaje seleccionado se agrega al recorrido que estamos construyendo y luego se vuelve a repetir este proceso a partir del PET destino del viaje recién agregado. Si se da el caso de llegar a un PET desde donde no partan viajes activos, lo que se agrega al recorrido es el viaje espera que parte desde dicho PET. De esta manera, repitiendo este procedimiento, iremos agregando viajes activos o viajes esperas en cada una de las iteraciones hasta llegar a un último PET que será aquel que no posea viajes activos ni esperas de salida. Lo que habremos conseguido es un recorrido sin Km. expresos. La función además de retornar el recorrido construido, retornará una lista con los viajes activos que no se usaron y que por lo tanto aún están disponibles. Esta nueva lista se calcula quitando los viajes activos, que fueron utilizados en el armado del recorrido, de la lista recibida al inicio como entrada.

- *ObtenerUnRecorrido_MetodoAleatorio*: Al igual que en la instrucción anterior, esta instrucción recibirá como entrada un conjunto de viajes activos que podremos utilizar. La dinámica de esta instrucción consiste en ir sorteando viajes activos del conjunto de viajes activos de entrada e ir agregándolos al recorrido a construir. En cada paso, y para determinar si puedo o no agregar el viaje sorteado al recorrido, se deberá comprobar si el viaje elegido es compatible con el resto de los viajes que ya contiene el recorrido. Si la respuesta es afirmativa el viaje es agregado y se elimina de la lista de viajes activos que están disponibles, en caso contrario el viaje no se agrega y se continúa con la próxima iteración. Este procedimiento se repite hasta que no

queden viajes activos disponibles o hasta que todos los viajes que quedan disponibles no sean compatibles con el resto de los viajes incluidos en el recorrido que armamos. Hasta aquí tenemos un recorrido formado exclusivamente por viajes activos. Lo que se hace a continuación para completar el recorrido es agregar aquellos viajes espera y expresos necesarios para que el recorrido construido sea factible.

- *CargoViajesAUtilizar_TodosLosPosibles*: Esta instrucción retorna como salida el conjunto con todos los viajes activos que están en el GG.

- *ArmarServicios_RecorridoEntero*: Esta instrucción recibe un recorrido factible como entrada y retorna como salida un nuevo servicio al cual le asoció el recorrido recibido.

- *ArmarServicios_RecorridoRecortadoEnPartes*: Esta instrucción recibe un recorrido factible como entrada y un entero N y retorna como salida un nuevo servicio al cual le asociará el recorrido recibido recortado en N partes iguales. Los nuevos recorridos obtenidos, producto de cortar el recorrido original, también deben ser factibles y cada uno de los viajes que estos contengan deben pertenecer al recorrido original.

- *ArmarServicios_MicroTurnos*: Esta instrucción tiene visibilidad sobre el MRL porque se precisa evaluar el cumplimiento de las reglas laborales y la idea es la de generar recorridos lo suficientemente cortos como para que se ajusten a la definición de microturno de manera de que tengan que respetar reglas laborales más holgadas. Esta instrucción también recibe por parámetro un recorrido factible. Lo que se hace es ir construyendo nuevos recorridos que respeten las reglas laborales. Se itera sobre cada uno de los viajes del recorrido original y se analiza si, agregar el viaje actual al nuevo recorrido que estamos construyendo, respeta las reglas laborales. Si esto se cumple, el viaje es agregado al nuevo recorrido y se sigue iterando sobre el recorrido original. Si por el contrario, agregar el viaje actual al nuevo recorrido hace que este no respete las reglas laborales, entonces se cierra este recorrido sin agregar el viaje en cuestión y se comienza uno nuevo. Este proceso se repite con el resto del recorrido original intentando armar nuevos recorridos que respeten las reglas laborales. Finalizado el proceso habremos cortado el recorrido original en N nuevos recorridos, donde N puede ser igual o mayor a cero. Si el número es mayor a cero, se retorna un nuevo servicio al cual se le asocian todos los recorridos construidos. En caso contrario se retorna null. Para hacer uso de esta instrucción deben definirse e inicializarse las variables *minMinutosTurno* y *maxMinutosTurno*.

- *ResetearCantidadIntentos*: Pone en cero la variable de nombre cantIntentos.

- *ImprimirInformacionServiciosArmados*: En esta instrucción se centraliza el manejo del despliegue de los datos que hacen al servicio. Los datos que imprime esta instrucción son la cantidad de vehículos asignados, la cantidad de turnos asignados, la cantidad de km. expresos, el costo en dinero que debemos

destinar en los turnos que quedaron armados y el detalle de los viajes que se realizan en cada uno de los servicios.

D.5.2. Condiciones

- *NoEstanTodosLosViajesActivosAsignados*: Esta condición recibe como entrada un conjunto de viajes activos y devuelve verdadero en caso de que este conjunto sea menor al conjunto formado por la totalidad de viajes activos que están definidos en el GG. Es decir, retorna verdadero en caso de que el conjunto recibido no coincida con todos los viajes activos del problema lo cual significa que, en dicho conjunto, no están todos los viajes activos. En caso contrario la condición devuelve falso. Esta condición trabaja además con dos variables globales de nombres *maxIntentos* y *cantIntentos*. Cada vez que se invoca esta condición se incrementa en 1 la variable *cantIntentos*. Si la variable *cantIntentos* supera el valor de *maxIntentos*, independientemente del contenido del conjunto de viajes, se devuelve falso.

D.5.3. Input

A continuación se especifican los distintos archivos de entrada definidos en Marco Polo para generar soluciones iniciales. Es importante destacar que un usuario con perfil programador podrá implementar sus propias técnicas para la generación de soluciones iniciales simplemente generando las nuevas instrucciones y/o condiciones que considere necesarias, junto con su correspondiente archivo de entrada en el cual deberá invocarlas para hacer uso de ellas.

D.5.3.1. Un recorrido por servicio

```
valueInt maxIntentos 80
CargoViajesAUtilizar_TodosLosPosibles
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_RecorridoEntero
if NoEstanTodosLosViajesActivosAsignados
jumpRel -3
ImprimirInformacionServiciosArmados
```

Con este archivo de configuración se arman servicios con un solo turno. Es decir que el recorrido del vehículo coincidirá con el recorrido del turno. A grandes rasgos, en esta configuración, se arma una iteración en donde cada ciclo busca un recorrido sin expresos para luego armar un servicio. Estas iteraciones se realizarán hasta que todos los viajes activos definidos en el GG hayan sido asignados o hasta que en un máximo de 80 intentos no se lograron cubrir todos los viajes.

Se define al comienzo el valor que tendrá la variable global *maxIntentos* y se invoca la instrucción *CargoViajesAUtilizar_TodosLosPosibles* con lo cual se indica que se trabajará a partir del conjunto completo de viajes activos del problema. Luego se invocan las instrucciones *ObtenerUnRecorrido_MetodoSinExpresos* cuyo retorno será la entrada de la siguiente función, *ArmarServicios_RecorridoEntero*, para obtener el servicio con el recorrido generado. A continuación la condición

NoEstanTodosLosViajesActivosAsignados evaluará si aún quedan viajes activos a utilizar o si, por el contrario, todos los viajes activos ya fueron utilizados. Si la evaluación de esta condición resulta verdadera, entonces la siguiente instrucción será ejecutada (*jumpRel -3*) con lo cual se volverá a ejecutar la instrucción *ObtenerUnRecorrido_MetodoSinExpresos* ya que el puntero de instrucción se moverá 3 lugares hacia arriba (Anexo E, Sección E.3.4). En caso en que la evaluación de la condición resulte falsa, el salto no será ejecutado y se pasará a ejecutar la instrucción ImprimirInformacionServiciosArmados. Una vez finalizada la impresión de pantalla el flujo es finalizado ya que no queda ninguna instrucción posterior a ser ejecutada.

D.5.3.2. N Recorridos por Servicio

```
valueInt maxIntentos 80
valueInt cantPartes 2
CargoViajesAUtilizar_TodosLosPosibles
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_RecorridoRecortadoEnPartes
if EstanTodosLosViajesActivosAsignados
jumpRel -3
ImprimirInformacionServiciosArmados
```

Este archivo de configuración es similar al anterior con la única diferencia de que en este caso se arman servicios con dos turnos. Para lograr esto se define la variable *cantpartes* a la cual se le fija el valor correspondiente a la cantidad de turnos que deseamos obtener por servicio y luego, al momento de armar dicho servicio, se invoca la instrucción *ArmarServicios_RecorridoRecortadoEnPartes*. Con esto indicamos que, no queremos simplemente que se agregue a un servicio el recorrido devuelto por la instrucción *ObtenerUnRecorrido_MetodoSinExpresos*, sino que queremos que además, previamente, éste sea dividido en la cantidad de recorridos que se indique en la variable *cantpartes*.

D.5.3.3. Un viaje por micro turno

```
valueInt maxIntentos 200
valueInt minMinutosTurno 1
valueInt maxMinutosTurno 60
CargoViajesAUtilizar_TodosLosPosibles
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_MicroTurnos
if EstanTodosLosViajesActivosAsignados
jumpRel -3
```

En este archivo de configuración, al igual que en el anterior, la idea base es iterar buscando recorridos para luego recortarlos de alguna manera y finalmente armar el servicio. En este caso la forma de armar el servicio será utilizando la instrucción *ArmarServicios_MicroTurnos*. Esta instrucción hace uso de dos variables las cuales son definidas como variables globales. La variable *minMinutosTurno* indica la cantidad mínima de minutos que puede durar un turno y, análogamente, la variable *maxMinutosTurno* indica la cantidad máxima. La idea, detrás de los valores fijados en este archivo en particular para dichas variables, es que los turnos generados contengan solamente un viaje.

D.5.3.4. MicroTurnosPorServicio

```
valueInt maxIntentos 80
valueInt minMinutosTurno 180
valueInt maxMinutosTurno 240
CargoViajesAUtilizar_TodosLosPosibles
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_MicroTurnos
if EstanTodosLosViajesActivosAsignados
jumpRel -3
ResetearCantidadIntentos
valueInt minMinutosTurno 120
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_MicroTurnos
if EstanTodosLosViajesActivosAsignados
jumpRel -3
ResetearCantidadIntentos
valueInt minMinutosTurno 60
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_MicroTurnos
if EstanTodosLosViajesActivosAsignados
jumpRel -3
ResetearCantidadIntentos
valueInt minMinutosTurno 1
ObtenerUnRecorrido_MetodoSinExpresos
ArmarServicios_MicroTurnos
if EstanTodosLosViajesActivosAsignados
jumpRel -3
```

En este archivo se pueden diferenciar varias etapas. En la primera, la idea es similar a la del archivo anterior, con la diferencia de que cada turno contendrá más de un viaje debido a los valores fijados para las variables *minMinutosTurno* y *maxMinutosTurno* (180 y 240 respectivamente). En esta etapa, entonces, se estarán buscando recorridos de turnos, que respeten las reglas laborales, con una duración de entre 180 y 240 minutos. Dadas estas restricciones de duraciones máximas y mínimas, es muy probable que en esta primera parte no se logren cubrir todos los viajes activos y debamos ser más flexibles a la hora de armar los siguientes turnos si es que queremos lograr cubrir todos los viajes. Pensando en esto es que, a continuación, se vuelve a cero la cantidad de intentos con la instrucción *ResetearCantidadIntentos* y luego se define un nuevo valor para la variable *minMinutosTurno*. En esta segunda ocasión se intentará armar turnos con una duración mínima de 120 minutos. Esta misma idea se repite dos veces más y aceptando cada vez una duración mínima más pequeña. Llegada la última instrucción es de esperar haber cubierto todos los viajes activos que el GG tiene definidos.

Anexo E. Workflow

E.1. Introducción

El módulo Workflow (en adelante WF) es genérico y puede ser utilizado en diversidad de situaciones. La motivación para crear este módulo fue la necesidad de obtener cierta flexibilidad y poder ejecutar controles sobre la secuencia de ejecución de cualquier proceso que lo requiera.

Un WorkFlow nos da la posibilidad de contar con muchas instrucciones y decidir cuales ejecutar y en qué orden. Se tiene la posibilidad de ejecutar cierta secuencia de ejecuciones una determinada cantidad de veces o en base a una evaluación. Mientras que las instrucciones son fijas, es decir, previamente definidas, el flujo en las que estas serán ejecutadas es configurable.

En Marco Polo definimos el módulo WF como una secuencia de acciones (instrucciones) que se hacen de forma consecutiva sobre un conjunto de datos. Una vez inicializados los datos de entrada, estos son procesados en cada una de las sucesivas etapas hasta que, en la última, se devuelven el o los resultados.

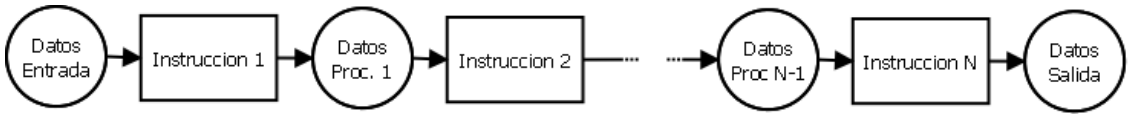

Figura E.1: Secuencia de instrucciones

E.2. Objetivos

Los objetivos del módulo WF son:

- Reflejar, mecanizar y automatizar un proceso
- Establecer mecanismos de control
- Independizar el flujo de la lógica en cada instrucción
- Facilitar la construcción y modificación de secuencias de instrucciones

E.3. Conceptos fundamentales

E.3.1. Datos

En los datos se encuentra información etiquetada. Se puede contar con cualquier cantidad de datos, toda la que se precise y cada uno de ellos será identificado con un nombre. Cada dato, asociado a un único nombre, tendrá un valor y este podrá ir cambiando en el transcurso del flujo por medio de las instrucciones. Además, el valor puede ser cualquier tipo de datos.

E.3.2. Instrucción

En cada una de las instrucciones estará definida la lógica a ser ejecutada. Aquí dentro se concentra el método de la instrucción que procesará, transformará, calculará, etc. los datos de manera explícita. Finalizada la ejecución se obtiene un resultado, es decir, un nuevo conjunto de datos. La instrucción puede agregar, modificar o quitar datos siendo importante que esta información sea bien conocida ya que con ella podremos saber si dos instrucciones son intercambiables o si una puede venir después de otra.

Una instrucción queda definida entonces con:

- Los datos de entrada: datos que se le deben ingresar a la instrucción para que esta opere correctamente
- La definición de la Instrucción: se indica que operaciones serán ejecutadas cuando la instrucción sea invocada
- Los datos de salida: cual o cuales datos serán retornados una vez finalizada la ejecución

E.3.3. Condición

La condición realiza una evaluación ya sea con los datos de entrada o con cualquier otra consideración que esta precise para dar una respuesta. Ejecutada la evaluación el resultado puede ser positivo o negativo (si o no) y en base a esto es si la siguiente instrucción del flujo será ejecutada o no respectivamente. De esta manera se evita un flujo lineal y se cuenta con la posibilidad de ejecutar instrucciones en base de condiciones.

Un ejemplo de un WorkFlow con una condición en el medio sería el siguiente:

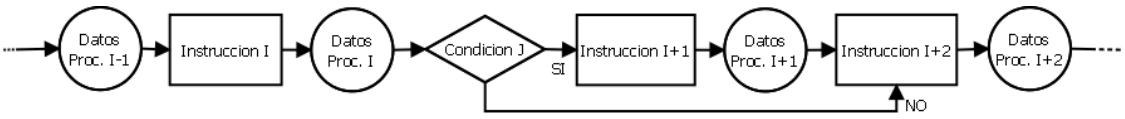

Figura E.2: Ejemplo de workflow

Si la condición J realiza la evaluación y el resultado es verdadero, la siguiente instrucción será la I+1 y es importante destacar que esta tendrá como datos de entrada a *Datos Proc. I*. Análogamente, si la condición J devuelve un resultado negativo, la siguiente instrucción a ejecutarse será la I+2 y de igual manera los datos de entrada serán *Datos Proc. I*.

E.3.4. Jump y JumpRel

Con *Jump* y *JumpRel* podemos realizar saltos y movernos entre diferentes instrucciones del flujo. Estas vienen acompañadas con un número entero. Para el caso del *Jump* el número indicará la posición absoluta en donde se encuentra la próxima instrucción a ser ejecutada mientras que, para el caso de *JumpRel*, el número se toma

como una distancia relativa y la próxima instrucción a ejecutar será la que se ubica a esa distancia del *JumpRel*. En la Figura E.3 se puede ver cómo trabajan el *Jump* y el *JumpRel*.

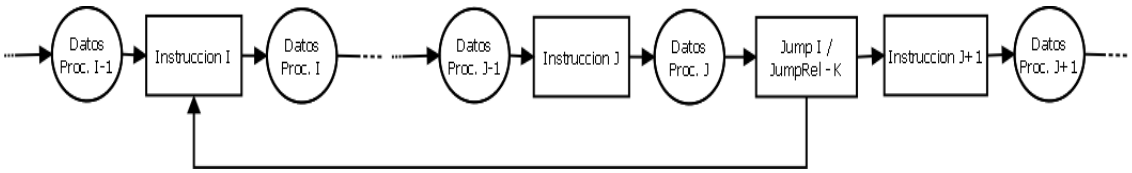

Figura E.3: Jump y JumpRel

En este caso, al igual que en el caso del condicional, los datos de entrada para *Instrucción I* serán los *Datos Proc. J*.

E.3.5. ValueInt y ValueString

Con estos dos conceptos podemos agregar datos al problema o inicializar los datos previamente definidos. Para poder utilizar estas funciones será necesario que el dato sea del tipo entero o texto. Por ejemplo, si existe un dato con el nombre *cantIntentos*, podemos querer poner el valor del mismo en cero y para ello se cuenta con *valueInt*. Como puede notarse, si se quisiera inicializar un dato en donde el tipo del valor sea otro distinto al de entero o texto se debería crear una instrucción para que se encargue de hacer este trabajo.

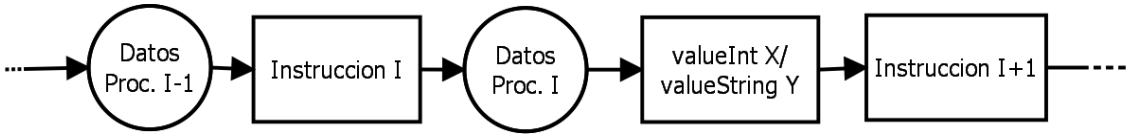

Figura E.4: ValueInt y ValueString

Nuevamente la instrucción *Instrucción I+1* recibirá como datos de entrada los *Datos Proc. I*.

E.3.6. Exit

Con el *exit* se corta la ejecución del flujo. En un flujo sin condiciones este concepto no tendría razón de ser ya que todas las instrucciones después del *exit* se podrían haber obviado desde un comienzo. Justamente, este concepto tendrá su mayor utilidad y sentido inmediatamente después de una condición.

E.4. Funcionamiento

La utilización del WF es muy sencilla y este módulo puede ser invocado desde cualquier parte del proyecto, incluso ser exportado a cualquier otro. Primero, se deben implementar las *Instrucciones* que se pondrán a ejecutar en el WF y las funciones que se utilizarán para evaluar *Condiciones*. Estos dos *Elementos* asumirán que reciben como entrada los datos que ellos utilizarán en su función. Luego de realizado el procesamiento la Instrucción devolverá o bien, los datos que recibió inicialmente transformados, o bien, datos nuevos. Por su parte, la condición también asume que

recibe los datos que esta precisa, pero a diferencia de la Instrucción, esta devuelve un valor booleano (true o false).

Luego de definidos estos elementos hay que decidir qué instrucción viene después de que otra, que condición se evaluará para determinar si una instrucción se debe realizar o no y que saltos relativos o absolutos se agregaran para determinar un control de flujo. Acomodando adecuadamente los elementos se puede conseguir un flujo con iteraciones. En definitiva estamos hablando de levantar el WF. Aquí es preciso tener en cuenta el orden en que se colocan cada uno de los elementos ya que cada uno de ellos debe proveer al siguiente los datos que este necesita recibir como entrada. Y sobre todo, armarlo con criterio para conseguir el fin que se quiere alcanzar.

Ahora el esqueleto está armado y, antes de hacer correr la máquina, se deben inicializar los datos de entrada, es decir, los datos que se le darán a la primera instrucción colocada en el flujo. Luego del primer procesamiento, el resultado se pasará al segundo elemento y así sucesivamente hasta que el flujo se complete. El resultado del flujo, serán los datos de salida de la última instrucción ejecutada en el flujo.

E.4.1. Ejemplo de la tostada

En el siguiente ejemplo se muestra como se puede aplicar el módulo WF para la sencilla tarea de preparar una tostada. Las instrucciones con las que contamos son: "*Tostar*" y "*Untar manteca*" y la condición utilizada será "*Suficientemente tostado*". Con estos elementos ordenados correctamente y contando con el dato de entrada correcto, en este caso "*pan*", podremos comer una tostada con manteca. El flujo sería el siguiente:

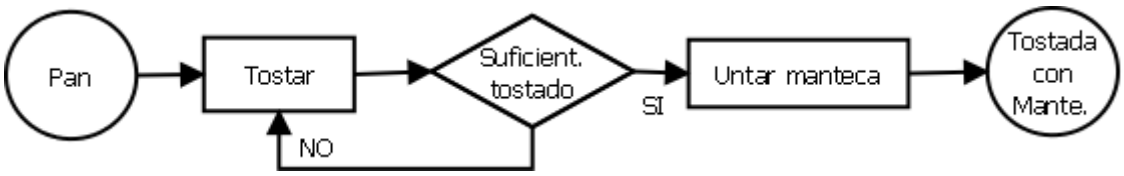

Figura E.5: Ejemplo de la tostada

Aclaración 1: los datos intermedios no están representados en la figura

Aclaración 2: existe un *jumpRel – 1* implícito para pasar de "*Suficientemente tostado*" a "*Tostar*" en caso de que la evaluación de la condición resulte negativa

El ejemplo es sencillo y útil. Sirve para hacernos entender el concepto global de un WorkFlow. Supongamos que queremos probar tostada con dulce de leche, sería cuestión de cambiar la instrucción "*Untar manteca*" por "*Untar dulce*" y si al untar la tostada la rompemos hay que mejorar la implementación de la instrucción. La condición "*Suficientemente tostado*" podría adaptarse con nuestra preferencia en cuanto a cuan tostado queremos el pan, cambiando su implementación para lograr una tostada más o menos tostada. Por último podríamos cambiar el dato de entrada para tostar otras cosas diferentes al pan, pero ejecutando siempre la misma estrategia.

E.5. Diseño e Implementación

E.5.1. Diagrama de clases

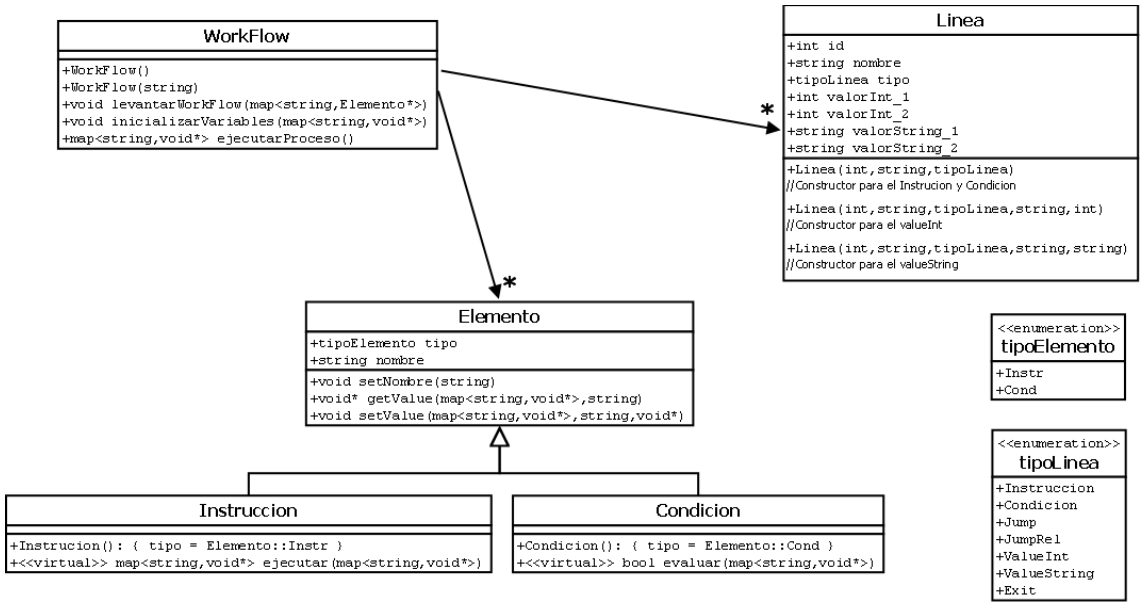

Figura E.6: Diagrama de clases

E.5.2. Controlador

La clase WorkFlow es el controlador del módulo WF. Sus tres funciones principales son:

- *levantarWorkFlow*: esta función recibe por parámetro todos los elementos que se utilizarán en el flujo
- *inicializarVariables*: esta función recibe por parámetro todos los datos que serán utilizados como entrada en el WorkFlow
- *ejecutarProceso*: inicia la ejecución del flujo y devuelve todos los datos de salida del mismo

El controlador contendrá dos listas durante toda la ejecución. Una de ellas contiene elementos de tipo *Elemento*, la cual servirá para almacenar todas las instrucciones y condiciones definidas. La segunda contiene elementos de tipo *Linea*, y es en ella donde se guardará la estructura del flujo.

E.5.3. Modo de Uso

A continuación se detallan los pasos que debería seguir un usuario programador para hacer uso del módulo WF

1. Implementar las instrucciones y condiciones que se van a querer usar en el flujo

Para el caso de las instrucciones se debe crear una clase que derive de *Instruccion*. Esta deberá asignar al atributo *nombre*, heredado de la clase *Elemento*, el nombre de la instrucción e implementar la función *ejecutar*. Esta función recibe como entrada los datos que se podrán utilizar en ella los cuales vienen en un diccionario (*map*) con clave *string* y valor *void**. La función *ejecutar* podrá obtener un dato en particular usando la función heredada *getValue*, a la cual se le debe pasar por parámetro el nombre del dato a obtener. Será responsabilidad del programador castearlo al tipo de datos que corresponda. El hecho de que los tipos de datos sean de tipo *void** es una ventaja ya que el flujo puede manejar cualquier tipo de datos. Se debe tener especial cuidado en que los datos que requiera la función *ejecutar* implementada sean luego provistos en el conjunto de datos de entrada que se le pasen a la función al ser invocada. El retorno de la función *ejecutar* es un diccionario con los todos los datos (los que se crearon, modificaron y/o eliminaron).

Para implementar las condiciones se debe seguir un procedimiento similar al de las instrucciones. Se debe crear una clase que herede de *Condicion*, asignar el nombre de la condición al atributo *nombre*, e implementar la función *evaluar*. Esta función debe devolver un valor booleano y podrá realizar su trabajo utilizando el diccionario de datos que recibe como entrada. Este diccionario se utiliza igual que en la instrucción con lo cual se deben tener las mismas consideraciones.

2. Crear el archivo de configuración donde se detalla el armado del flujo

La parametrización de cómo armar el flujo está detallada en un archivo de configuración. Cuando se crea la clase WorkFlow, la ruta de donde se encuentra este archivo y su nombre son pasados como parámetro, y de esta manera el controlador carga sus variables e instancia todas las clases necesarias. El archivo se organiza en líneas y, en cada línea, se indica un *Elemento*. Los elementos pueden ser:

- *Instruccion*: deben pertenecer al conjunto de instrucciones que se implementaron en el paso 1. Para hacer referencia a un elemento de tipo *Instruccion* se debe escribir en la línea el nombre de la instrucción implementada

- *Condicion*: al igual que con las instrucciones, las condiciones que aparezcan en el archivo de configuración deben pertenecer al conjunto de condiciones implementadas en el paso 1. Para hacer referencia a un elemento de tipo *Condicion* se debe escribir la palabra *if* seguida de un espacio y el nombre de la condición implementada

- *ValueInt*: con este elemento se puede inicializar un dato del tipo entero, escribiendo la directiva *ValueInt* seguido de un espacio, el nombre del dato, otro espacio y el valor entero a ser asignado

- *ValueString*: con este elemento se puede inicializar un dato del tipo cadena de caracteres, escribiendo *ValueString* seguido de un espacio, el nombre del dato, otro espacio y la cadena de caracteres a ser asignada

- *Jump*: sirve para indicar cuál será el siguiente elemento a ejecutar indicando una posición absoluta en el archivo. Se debe escribir Jump, seguido de un espacio y la posición absoluta del próximo elemento

- *JumpRel*: sirve para indicar cuál será el siguiente elemento a ejecutar indicando la posición relativa del mismo. Se debe escribir JumpRel, seguido de un espacio y la posición relativa del próximo elemento

- *Exit*: escribiendo la directiva *Exit* se indica que finaliza la ejecución del flujo

En el ejemplo de la tostada el archivo de configuración tendría que contener las siguientes líneas:

```
Tostar
If
notSuficientementeTostado
jumpRel -2
UntarManteca
Exit
```

Figura E.7: Ejemplo de archivo de configuración del WF

3. Instanciar el WorkFlow

La clase que vaya a utilizar el módulo WF debe hacer lo siguiente:

- Incluir el archivo de cabeceras *WorkFlow.hh*

- Hacer un *new* para cada elemento (instrucciones y condiciones) que se van a utilizar en el flujo y agregarlas a la lista que maneja el controlador para este propósito

- Ejecutar la función *levantarWorkFlow* y pasarle por parámetro la lista cargada anteriormente

- Crear todos los datos que se pasarán como entrada al flujo a ejecutar e insertarlos en un diccionario con una clave que los identifique

- Ejecutar la función *inicializarVariables* y pasarle por parámetro el diccionario creado anteriormente

- Ejecutar la función *ejecutarProceso* y en el retorno de dicha función estará el resultado arrojado por el flujo

E.6. WorkFlow interactivo

La implementación del módulo WF permite la posibilidad de ejecutarlo desde consola y hacer que se vaya ejecutando un flujo dinámicamente. Las líneas se van ingresando por consola escribiendo cada elemento a ser ejecutado. La única restricción en este tipo de ejecución es que no se pueden realizar saltos hacia delante por razones obvias.

Para los casos en que se realice un salto hacia atrás el proceso ejecutará todos los elementos que aparecen después de realizar el salto hasta llegar al elemento posterior a dicho salto, en ese momento el WorkFlow vuelve a solicitar la siguiente instrucción a ejecutar.

Referencias

1. **Desaulniers, G. and Hickman, M. D.** Chapter 2: Public Transit. [book auth.] C. Barnhart and G. Laporte. *Handbooks in OR and MS, Vol.14.* s.l. : Elsevier B.V., 2007.

2. **Bello, F., Machado, M. and Vignola, J.** *Una Metaheurística para el Problema de Ordenamiento de Flota y Asignación de Tripulación en Sistemas de Transporte Público.* Montevideo : s.n., 2010. Proyecto de Grado.

3. **Chicano García, J. F.** *Metaheurísticas e Ingeniería del Software.* Lenguajes y Ciencias de la Computación, Universidad de Málaga. Málaga : s.n., 2007. Tesis Doctoral.

4. **Goldberg, D. E.** *Genetic Algorithms in Search, Optimization and Machine Learning.* New York : Addison-Wesley, 1989. 0-201-15767-5.

5. **Lourenço, H. R., Paixão, J. P. and Portugal, R.** *Multiobjective metaheuristics for the bus-driver scheduling problem.* Barcelona : s.n., 2001.

6. **Ball, M., Bodin, L. and Dial, R.** *A matching based heuristic for scheduling mass transit crews and vehicles.* s.l. : Transportation Science Vol. 17, No. 1, 1983.

7. **Freling, R., Huisman, D. and Wagelmans, A. P. M.** *Models and Algorithms for Integration of Vehicle and Crew Scheduling.* Rotterdam : Journal Of Scheduling 6, 2003. 63-85.

8. **Gaffi, A. and Nonato, M.** An integrated approach to the extra-urban crew and vehicle scheduling. [book auth.] Wilson N.M.H. (ed.). *Computer-Aided Transit Scheduling.* Berlin : Springer, 1999.

9. **Huisman, et al.** *Multiple-Depot Integrated Vehicle and Crew Scheduling.* Rootterdam : Transportation Science, Vol.39, No.4, 2005.

10. **Hao, J. K. and Lauren, B.** *Simultaneous Vehicle and Crew Scheduling for Extra Urban Transports.* Berlin : Springer-Verlag, 2008.

11. **Papadimitriou, C. H. and Steiglitz, K.** *Combinatorial Optimization: Algorithms and Complexity.* s.l. : Prentice-Hall, 1982. 0-13-152462-3.

12. **Luna Valero, F.** *Metaheurísticas avanzadas para problemas reales en redes de telecomunicaciones.* Málaga : Universidad de Málaga, 2008.

13. **Glover, F. and Kochenberger, G. A.** *Handbook of Metaheuristics.* New York : Kluwer Academic Publishers, 2003. 1-4020-7263-5.

14. **Holland, J. H.** *Adaptation in natural and artificial systems.* Ann Arbor : University of Michigan Press, 1975.

15. **Garey, M. R. and Johnson, D. S.** *Computers and Intractability - A Guide to the Theory of NP-Completeness.* s.l. : Freeman, W.H., 1979.

16. Mallba Library. *NEO Networking and Emerging Optimization.* [Online] [Cited: Febrero 16, 2013.] http://neo.lcc.uma.es/software/mallba/index.php.

17. MPICH. *High-performance and Portable MPI.* [Online] [Cited: Febrero 16, 2013.] http://www.mcs.anl.gov/research/projects/mpich2.

18. mini-XML. *Lightweight XML Library.* [Online] [Cited: Febrero 16, 2013.] http://www.minixml.org/.